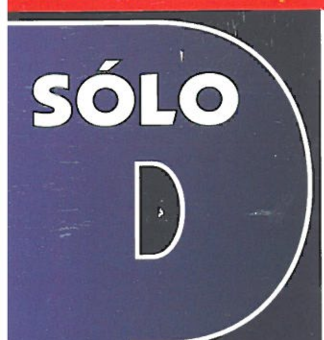
LA PRIMERA REVISTA DE PROGRAMACION EN CASTELLANO



PROGRAMADORES



Revista especializada para usuarios de PC

AÑO 3. Nº 25
250 PTAS.
ARGENTINA 9'50 $
CHILE 3000 $
PORTUGAL 1500$

LINUX: PRESENTACIÓN DE TEXTO EN PANTALLA

WEB: DESCRIPCIÓN AVANZADA DE LA INTERFAZ CGI

VISUAL BASIC: CONCEPTOS DE PROGRAMACIÓN

CD-ROM DE REGALO: GESTOR DE BASES DE DATOS 4 TH DIMENSION

Y además:

- La BIOS del PC
- Cómo programar una demo
- Redes: Protocolos TCP/IP

CURSOS PRÁCTICOS DE:

- Programación
- Grandes Sistemas
- Visual C++ 4.0
- Modos X de la VGA

TÉCNICAS DE COMPRESIÓN

ALGORITMO HUFFMAN

8 413042 303299 00025



Network Management

La tarjeta de respue[illegible] ya ha sido utilizada.

Unix

Parallel Systems

Cliente/Servidor

Internet

Bases de datos relacionales

Objetos

N W E S

NO se preocupe. Envíenos por correo electrónico su nombre, dirección y función a:
SDOEMEA@UK.IBM.COM
o llámenos al 900 100 400

Cómo lograr y mantener una posición líder en el vertiginoso negocio del desarrollo de software.

Para desarrollar programas se necesitan buenas ideas así como los conocimientos y las herramientas necesarios para tranformarlas en soluciones excepcionales. La clave para llegar y mantenerse en una posición destacada es el acceso fácil a la información necesaria. Por eso IBM ha creado D_Mail: un nuevo servicio de información para empresas y profesionales desarrolladores de software.

D_Mail le facilita la más reciente información sobre el impacto potencial de las últimas innovaciones, análisis detallados sobre tecnologías clave, el panorama de oportunidades profesionales para el mundo de las soluciones de software y mucho más. ¡Sin coste para usted!

Sólo tiene que devolvernos cumplimentada la tarjeta adjunta, y nosotros le ayudaremos a trazar el rumbo correcto hacia el futuro.



Soluciones para nuestro pequeño mundo



SEPTIEMBRE 1996. Número 25
SÓLO PROGRAMADORES es una publicación de Tower Communications

Editor
Antonio M. Ferrer Abelló
Director Editorial
Mario de Luis García
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
Director de Distribución
Enrique Cabezas García

Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador Técnico
Eduardo Toribio Pazos
Edición
Miguel Cabezuelo
Colaboradores
Fernando de la Villa, Fernando J. Echevarrieta, Pedro Antón. Juan M. Martín, Luis Martín, José M. Peco, José C. Remiro, Agustín Guillén, Daniel Navarro, Jorge del Río, Enrique De Alarcón, David Aparicio, M. Jesús Recio, Santiago Romero, Miguel Cubas, Vicente Cubas, Carlos Pérez

Jefe de Diseño
Fernando García Santamaría
Maquetación
Clara Francés
Tratamiento de Imagen
María Arce Giménez
Suscripciones
Isabel Bojo
Publicidad en Madrid
Erika de la Riva
Redacción, Publicidad y Administración
C/ Aragoneses, 7
28100 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86
Publicidad en Barcelona
Pepín Gallardo
(93) 213 42 29
Filmación
Filma Dos S.L.
Impresión
G. Reunidas
Distribución
SGEL

Depósito legal: M-26827-1994
ISSN: 1134-4792



EDITORIAL

ENVUELTOS POR LA RED

En los últimos tiempos, la popularización creciente de Internet como un fenómeno impulsado desde los medios de comunicación ha creado una necesidad impensable hasta hace poco: la comunicación electrónica para el gran público.

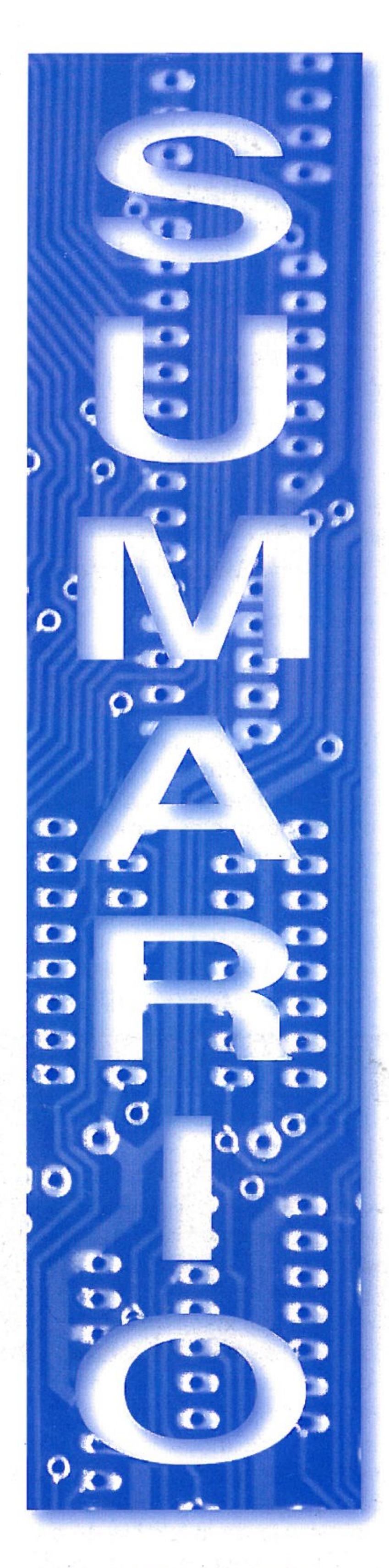
El nacimiento de empresas que ofrecen conexión a la Red de Redes ha tratado de cubrir con mayor o menor éxito una demanda de servicios como correo electrónico personal, suscripciones a noticias por tablón, navegadores multimedia y otros servicios novedosos. El paradójico problema para estas pequeñas empresas es que tienen dificultades para asimilar el éxito: Cuando el número de clientes empieza a aumentar, la complejidad de las instalaciones necesarias y la lentitud de los módems analógicos actuales produce una inevitable degradación en el servicio que pocos proveedores pueden resolver. Entonces se produce un fenómeno de "migración" masiva de la clientela de un proveedor a otro, buscando mejores precios y servicios, hasta que el ciclo se repite. Esto indica que se necesita una colaboración entre las grandes compañías de telecomunicaciones y los proveedores.

La lucha por conducir el mercado de la "aldea global" no ha hecho más que comenzar. Compañías como la recién llegada NetScape o el dominador Microsoft pujan por monopolizar el software que habrá de conducirnos por estas nuevas tecnologías. El concepto de que podemos ejecutar nuestro procesador de textos favorito sin que realmente se encuentre almacenado en el disco duro modificarán el concepto del software que hoy se tiene. Por un lado, el "alquiler" de la ejecución de programas frente a la compra del producto que hoy se realiza, obligará al usuario a pagar cantidades pequeñas pero continuas, a cambio de conseguir ventajas tales como actualizaciones instantáneas del software. El hecho de "pagar por usar" predice la llegada de una sociedad de consumo extrema, donde la gran multinacional puede controlar hasta el límite el uso que se hace de sus productos, de forma instantánea. Por otro lado, la piratería informática previsiblemente pasará de la inocente duplicación de software de hoy al acceso ilegal a los centros de información y servicios de la red del mañana, creando así inquietantes mafias organizadas.

Muchos nos preguntamos si la explosión de Internet, salvaje y libre hoy, no derivarán hacia el desarrollo de una red de comunicaciones "descafeinada", domesticada por los intereses de ciertas superpotencias mundiales y sus influyentes multinacionales, una red suavizada donde el individuo pasaría a ser un espectador más o menos pasivo. Ya hoy en día países como Francia o Estados Unidos intentan impedir la libertad de expresión y siguen una política de "policía de la red". La capacidad que ahora existe, donde un simple individuo puede actuar como emisor de información (anárquico, en el mejor sentido del término) frente a un público importante, puede ser un espejismo que desaparezca en menos de una década.

Hoy somos libres para manejar el timón de nuestro barquito cuando navegamos por Internet. Pero ¿y mañana?. Quizá nos encontremos dentro de un patinete de parque de atracciones, que sólo puede seguir su camino sobre sus tristes rieles. Más espeluznante aún puede ser imaginar grupos de personas que se reunan para observar el parque de atracciones desde el exterior del vallado, sin poder pagar una entrada.

Si la industria encuentra el modo de hacer rentable el cambio global del transporte informático que parece despuntar, entonces la posibilidad de que nuestro PC desaparezca para ceder su lugar a un nuevo televisor interactivo tomará cuerpo sólo por decisión nuestra. Es el momento de meditar.

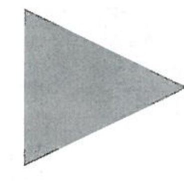




NOTICIAS:

NOVEDADES DEL MERCADO

Espacio destinado a informar al lector de las últimas novedades acontecidas en el mundo de la informática y el comentario de los libros de interés.





TECNOLOGÍA WEB (X):

LA INTERFAZ CGI

Tras una primera toma de contacto con los conceptos fundamentales asociados a la programación CGI, ahora se terminarán de exponer las facilidades.





CURSO DE PROGRAMACIÓN (XIX):

EVALUACIÓN DE EXPRESIONES

Los árboles son una buena estructura de datos para representar y evaluar expresiones.





SISTEMAS ABIERTOS (XIV):

ARQUITECTURA Y PROTOCOLOS TCP/IP

En este artículo se presenta la familia de protocolos asociados a la arquitectura TCP/IP.





GRANDES SISTEMAS (XV):

OBJETOS NATURAL (y III)

Se cierra con esta entrega el tema de los objetos natural, mediante el cual su autor ha descrito las características de cada uno.

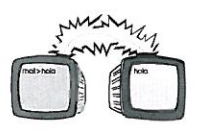




REDES LOCALES (V):

PROTOCOLOS TCP/IP

Las conexiones a Internet se han puesto tan de moda que el conocimiento de los protocolos TCP/IP se hace practicamente imprescindible.

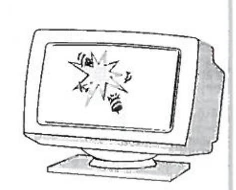




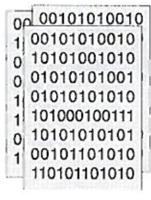
CURSO DE DEMOS (XV):

EL RELLENADO SÓLIDO

Este mes se tratará uno de los objetivos de este curso: aprender cómo hacer un rellenado sólido.







PC INTERNO (II):

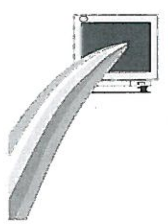
PROGRAMACIÓN DE LA ROM BIOS (II)

En esta ocasión se estudiarán a fondo todos sus servicios.

50 MODOS X DE LA VGA (IV):

RESOLUCIONES VIRTUALES

Una vez visto el tema de los registros, se inicia el estudio de las resoluciones de pantallas, que permitirán hacer scrolles multidireccionales.



56 SISTEMA OPERATIVO LINUX (XVIII):

LA LIBRERÍA CURSES

Existe una librería estándar en UNIX que permite realizar vistosas presentaciones de texto en pantalla aunque no se opere en modo gráfico.



62 TÉCNICAS DE COMPRESIÓN

ALGORITMOS HUFFMAN

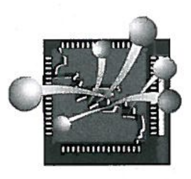
Para muchos, la compresión de datos tiene algo de esotérico, ¿cómo meter en un disco de 1.44Mb un programa que ocupa 3Mb?.



67 CURSO DE VISUAL BASIC 4.0 (VII):

TIPOS DE DATOS

De todas las etapas de la realización de una aplicación, la elaboración del código es, quizás, la que requiere un mayor nivel de conocimientos.



73 VISUAL C++ (VIII):

EL CONTROL TREECTRL

La encapsulación bajo clases es uno de los puntos fuertes de Visual C++ 4.0 y las MFC. En este artículo se detallarán en profundidad.



79 CONTENIDO DEL CD-ROM:

4TH DIMENSION

Este mes se incluye un CD-ROM con las versiones de evaluación para Windows de 4D y 4D Server. Se trata de un sistema gestor de bases de datos relacionales cliente/servidor de 32 bits con un atractivo interfaz gráfico



81 CORREO DEL LECTOR:

DUDAS DE LOS LECTORES

Espacio dedicado a resolver los problemas surgidos a los lectores en diversos aspectos de la programación.



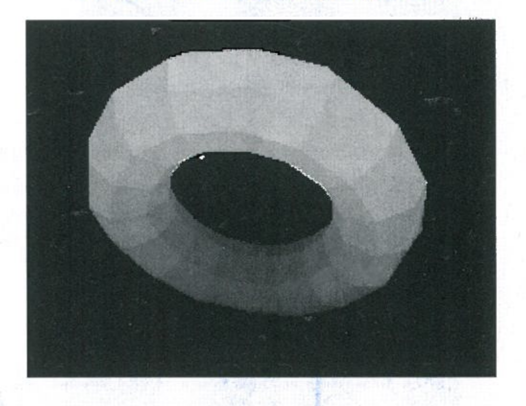
TÉCNICAS DE COMPRESIÓN (Pág 62)



ALGORITMO HUFFMAN

En este número de Septiembre viene a nuestra portada un interesante estudio sobre la compresión por códigos Huffman. Ya han acabado algunos de nuestros cursos habituales como el de Compiladores, pero desde aquí anunciamos otros nuevos que se incorporarán los próximos meses

REDES LOCALES (Pág. 34)



PROTOCOLOS TCP/IP

Estando tan de moda las comunicaciones, y en especial Internet, no se podía dejar pasar la ocasión de ofrecer un estudio de la familia de protocolos TCP/IP, cuyo es imprescindible, eso sin citar su importancia en entornos de redes locales. También se destaca nuestro Curso de Demos, donde por fin ofrecemos la esperada rutina de rellenado de polígonos.

NUEVO MODELO DE PROCESADOR
Intel anuncia el procesador Pentium a 200 MHz

Intel Corp. ha anunciado recientemente su nuevo modelo de procesador Pentium a 200 Mhz, que dotará los ordenadores de sobremesa durante esta segunda mitad del año. Con este modelo, Intel amplía la gama alta de sus procesadores, cuyas versiones a 166 y 150 Mhz fueron presentadas a principios de este mismo año. Este anuncio coloca ahora sus modelos a 120 y 133 Mhz dentro de la gama baja de procesadores.

Este nuevo procesador, basado en la tecnología de proceso avanzada de 0.35 micras de Intel, está en producción limitada, y la producción en volumen crecerá en los dos próximos trimestres. Este procesador está montado en un nuevo encapsulado PPGA (*Plastic Pin Grid Array*) más eficaz térmicamente. Esta nueva concepción, combinada con la avanzada tecnología de proceso, permite al procesador alcanzar sus altas tasas de velocidad de reloj. Para más información, dirigirse a la página Web de Intel en la dirección ***http:// www.intel.com/press-room***

64 BITS PARA WINDOWS NT"CAIRO"
Microsoft incorporará soporte para datos de 64 bits en Windows NT

Microsoft Ibérica ha anunciado que su sistema operativo Windows NT soportará datos de 64 Bits, con el objetivo de satisfacer las necesidades de los clientes que necesitan acceder de forma eficaz a grandes bases de datos. Esta funcionalidad está previsto que esté disponible sobre Windows NT "Cairo", e inicialmente será soportada sobre las plataformas Alpha de Digital.

Actualmente, Windows NT soporta aplicaciones de hasta 2 Gygabytes, suficiente para la mayoría de las soluciones corporativas de gama alta.

LANTASTIC v7.0
Artisoft Inc. lanza una nueva versión de su sistema de redes de área local

Artisoft acaba de lanzar la versión 7.0 de su sistema operativo de redes LANtastic, la primera red de área local preparada para Internet y específicamente diseñada para la pequeña y mediana empresa y grupos de trabajo. LANtastic v7.0 permite a todos los usuarios de la red compartir simultáneamente un único módem, línea de teléfono y conexión Internet.

Este software combina las nuevas opciones de conectividad a Internet y WAN con prestaciones de velocidad muy mejoradas, así como una significativa mejora en el manejo de la memoria y en las posibilidades de comunicación.

COMPUSERVE DESDE WINDOWS 95
Se incluirá acceso directo a Compuserve

Microsoft Corporation y Compuserve han alcanzado un amplio acuerdo que incluye, entre otras cosas, la inclusión de un acceso directo en Windows 95 a los servicios Online de Compuserve, al igual que el uso de Microsoft Internet Explorer como explorador por parte de Compuserve.

Los usuarios de futuras ampliaciones de este sistema operativo contarán en su escritorio con un icono de acceso directo a los servicios Online Compuserve Information Server y WOW!. Compuserve, por su parte, ofrecerá y recomendará a sus abonados la utilización del explorador de Microsoft para navegar por Internet.

El explorador estará incluido por defecto en los programas de instalación de los servicios Online de Compuserve anteriormente mencionados, así como en las ofertas de paquetes de software de Compuserve.

REALIDAD VIRTUAL

Modelo de realidad virtual de Stonechemge en Internet

English Heritage, Intel Corp. y otros líderes en el campo de la realidad virtual (VR) basada en la arquitectura Intel han desvelado los modelos VR de Stonehenge desarrollados gracias al procesador Pentium Pro de Intel. A partir de ahora, los cibernautas de todo el mundo podrán explorar un modelo 3D, científicamente exacto, del lugar megalítico inglés de Stonehenge desde un PC basado en Pentium Pro conectado al servidor corporativo de Intel en Internet.

El modelo funciona en los PCs de arquitectura Intel, y se recomienda el procesador Pentium más rápido posible para tener una experiencia más realista. Para ejecutar el modelo, los usuarios pueden descargar una copia gratuita de Viscape 3D de Superscape para Netscape Navigator 2.0 o versiones posteriores desde el Web de Intel. El modelo estará disponible en octubre. Para más información consultar la dirección *http://www.intel.com/pressroom.*

WEB AUTHORING MULTIPLATAFORMA

Claris lanza su creador de páginas Web

Como parte de su estrategia para ofrecer aplicaciones "fáciles de usar", Claris Corporation ha anunciado el lanzamiento de *Claris Home Page*, una aplicación de *web authoring* multiplataforma que correrá bajo Windows 95, Windows NT y Mac OS (en sistema 7.1 o superior, 680x0 y Power Macintosh y compatibles). Este nuevo lanzamiento ofrece una creación de páginas WYSIWYG y automatiza gran parte de las tareas más consumidoras de tiempo que conlleva el desarrollo de éste tipo de páginas.

Para los usuarios noveles en la creación de páginas Web, Claris Home Page "esconde" la complejidad de creación del código HTML, y lo genera automáticamente simplemente introduciendo el texto como en un tratamiento de textos y seleccionando ítems de la paleta de herramientas o desde los menús desplegables. Del mismo modo, para los usuarios más avanzados que quieran editar sus propios códigos para *authoring* más sofisticado, Claris Home Page ofrece ésta posibilidad y mucho más. Estos usuarios pueden también desarrollar sitios dinámicos e interactivos llamados *applets* (utilizando Java o JavaScript) y aplicaciones CGI.

SERVIDORES DE WEB DE LOTUS

Estrategia de Lotus para Internet basada en Domino

Dentro de su nueva estrategia para el entorno Internet, Lotus Development ha anunciado su servidor de web para aplicaciones interactivas basadas en Domino. Esta familia de productos de nueva generación se denominará Domino II. En base a los planes de entrega previstos, Lotus se concentrará en desarrollar rápidamente esta tecnología para conseguir un servidor web, basado en estándares Internet, así como una línea de servidores, clientes y herramientas que saquen partido de la estructura de Notes. Por otra parte, Lotus ha anunciado también la disponibilidad de la beta de Notes 4.5, basada también en la tecnología de Domino, que combina en un único cliente la mensajería de Lotus, su soporte para usuarios móviles y acceso directo a la red. Así mismo, soportará también los módulos Java, módulos compatibles con Netscape y aplicaciones de *groupware*, entre otras.

DELPHI LANGUAJE PACK Y TRASLATION SUITE

Borland anuncia nuevas herramientas de traducción para Delphi

Borland International Inc. ha anunciado recientemente el lanzamiento de Delphi Languaje Pack y Delphi Translation Suite, dos nuevas herramientas para la traducción de aplicaciones Delphi. Estos nuevos productos proporcionan una forma integrada y potente de traducción de aplicaciones Delphi hacia idiomas como el danés, holandés, alemán, francés, italiano y español, entre otros. Según fuentes de Borland, estas utilidades de programación usan la misma tecnología que utiliza la compañía para la conversión de sus productos hacia otras lenguas, ventaja que podrán utilizar los programadores en Delphi para distribuir sus desarrollos en diferentes idiomas. Las dos nuevas herramientas incluyen, entre otras cosas, traducción de mensajes de sistema para las librerías VCL y el Borland Database Engine (BDE), así como traducción de mensajes en los formularios, cajas de diálogo y menús, disponibles en las dos versiones. Además, Delphi Translation Suite incluye unas potentes funciones para automatizar el proceso de traducción. Para más información consultar la página web de Borland en la dirección *http://www.borland.com.*

IBM DB/2
Orientación a Objetos para DB/2

IBM continúa dotando a su base de datos relacional DB/2 de mayores contenidos multimedia y tecnología orientada a objetos, después de dar a conocer las nuevas extensiones multimedia que han sido incluidas en una de las bases de datos de más prestigio del mercado. Estas extensiones permitirán gestionar, manipular y distribuir eficazmente complejos datos multimedia compuestos de textos, imágenes, vídeo y audio en lo que representa la culminación de una estrategia dirigida a dotar de características de orientación a objetos de DB/2.

Disponibles para servidores y clientes basados en las plataformas operativas AIX, OS/2 y Windows NT, estas soluciones constituyen el componente clave de aproximación a la tecnología orientada a objetos de IBM en bases de datos, una iniciativa muy relacionada con las propuestas de sus principales competidores para lanzar servidores basados en tecnología orientada a objetos.

Por otro lado, la firma también anunció la disponibilidad de la versión 1.2 de DB/2 Parallel Edition, que ofrece a los usuarios una mejora aproximada del 50 por ciento en la relación de rendimientos de *queries* y de un 200 por ciento en la velocidad del proceso de transacciones.

VISUAL C++ 4.1
Nueva versión del compilador C++ de Microsoft

Microsoft ha anunciado recientemente el lanzamiento de la nueva versión 4.1 de su compilador Visual C++. Esta nueva revisión incluye soporte para Microsoft Internet Information Service, nuevos controles ActiveX (conocidos anteriormente como Controles OLE) e incremento de la ejecución en la gestión de grandes proyectos. Visual C++ 4.1 permite desarrollar componentes reutilizables a través de AppWizards pesronalizados, controles ActiveX y Mfc 4.1. Esta actualización incluye coporte para Windows 95 y NT.

Entre otras novedades, la nueva versión incluye también MFC for ActiveX Servers para la creación de aplicaciones Web interactivas usando el Microsoft Internbet Server API (ISAPI) y una extensión Wizard ISAPI para crear extensiones y filtros para servidores de Internet.

Los usuarios registrados de Visual C++ recibirán automáticamente la nueva versión. Para los nuevos usuarios, el precio del paquete rondará las 65.000 pesetas. Para más información consultar la dirección Internet http://www.microsoft.com/devonly.

JUST -IN-TIME
Compilador para Java de Symantec

Symantec ha anunciado la disponibilidad de su compilador Just-In-Time para Java, de forma gratuita, para los usuarios de Café. Este compilador transforma los *bytescodes* de Java en instrucciones de procesador nativo, para lograr una mayor rapidez de ejecución bajo 32 bits. El resultado de este proceso es una mejora de hasta 23 veces superior en la velocidad de ejecución de los *applets* Java y aplicaciones estándar. Al proporcionar Just-In-Time, Symantec posibilita una ejecución con velocidad similar al código nativo C/C++ para los *applets* web y las aplicaciones estándar de Java.

El nuevo compilador de Symantec está disponible en la dirección *http://cafe.symantec.com* para todos los compradores iniciales de Symantec Café.

PUENTE ENTRE JAVA Y C++
Nueva tecnología Java de ILOG y SunSoft

ILOG Inc. y SunSoft han anunciado un acuerdo para desarrollar un puente entre Java y C++. El proyecto, conocido como "TwinPeaks", cimentará el camino para conseguir una tecnología que soporte la totalidad de los componentes estándar C++ y ANSI C en Java. Este puente será un programa "pasarela" que permitirá a Java usar y reconocer los componentes C++ a través de un interfaz generado automáticamente.

Los componentes C++ están formados por dos elementos: un interfaz o fichero de cabecera y el código objeto. El "puente TwinPeaks" podrá leer y analizar estos ficheros de cabecera y creará todo un interfaz Java y un pequeño fragmento de código Java/C++ que convertirá las llamadas API y los formatos de datos de los componentes existentes. A través de esta conversión, Java estará capacitado para reconocer y usar los componentes C++ existentes.

Para más información:
http://www.ilog.com

VISUAL J++
Versión Beta del esperado compilador Java de Microsoft

Ya se encuentra disponible en Internet una versión Beta de la más esperada herramienta de programación Java, el nuevo Visual J++ de Microsoft. Java es el lenguaje más potente para el desarrollo profesional de aplicaciones Web actualmente. Basado principalmente en un subconjunto de funciones C++, Java proporciona un rápido desarrollo y una amplia portabilidad de las aplicaciones. Este nuevo producto incluye el conocido entorno de desarrollo de Microsoft, un debbuger gráfico, editor visual, un rápido compilador de código fuente, un completo tutorial *on-line*, orientación a objetos y una larga lista de herramientas para crear potentes páginas y aplicaciones Web de forma rápida y sencilla, definiendo de esta forma un nuevo estándar Java en J++, como ya ocurriera anteriormente con C y C++.

Entre las ventajas que ofrece este paquete destacan el soporte de applets Java, el desarrollo de aplicaciones en 32 bits para Windows 95 y Nt y la velocidad de compilación, que llega al millón de líneas por minuto. Además, Visual J++ eleva el lenguaje Java a un nivel más alto con la incorporación de controles ActiveX, incluyendo la posibilidad de creación de dichos controles. Tanto la versión Beta de Visual J++ como toda la información acerca de este producto están disponibles en la dirección ***http://www.microsoft.com/visualj.***

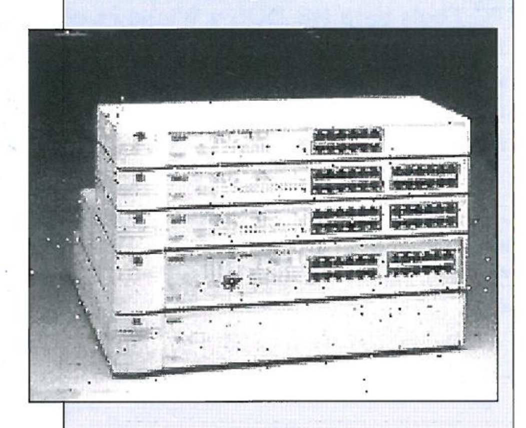
ESTRATEGIA DE TOPWARE ESPAÑA
El mejor software al mejor precio

Topware España ha desvelado su estrategia de ofrecer "El mejor software al mejor precio", respondiendo a la actual demanda del mercado, donde los precios son desorbitados. Para ello ofrece software de calidad a un precio asequible para todo el mundo, una filosofía de empresa que convence tanto a los emisores de licencias como a los clientes. Al mismo tiempo, Topware prescinde de las envolturas lujosas y distribuye el software en CD, con lo que los comercios podrán exhibir más Cds por metro cuadrado en sus escaparates. Entre la oferta de Topware se encuentran títulos como PC Tools para MS-DOS y Windows en inglés y castellano, el Antivirus Norton para los mismos entornos operativos y Graphics Works y Designer de Micrografx.

NOVEDADES

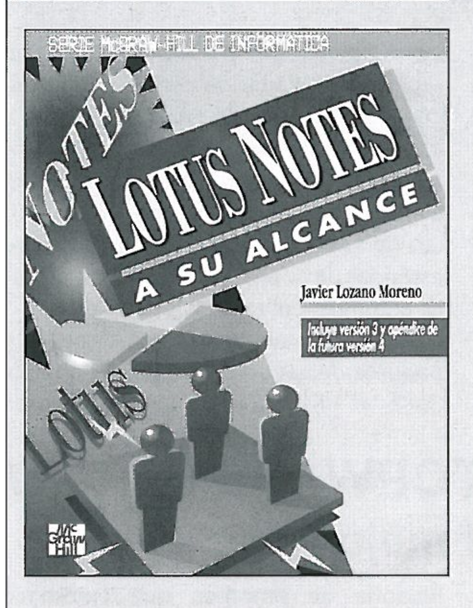
SuperStack II de 3COM

3COM ha lanzado al mercado el sistema apilable SuperStack II. Construido aprovechando el éxito de SuperStack, esta nueva versión se caracteriza por sus capacidades de escalabilidad, integración y gestión propias de los sistemas de red basados en chasis. Y es que la nueva versión introduce nuevas innovaciones tecnológicas antes no disponibles en sistemas apilables, como una completa gestión integrada de red basada en gráficos, capacidad de cambio en caliente y suministro de alimentación ininterrumpida.



Perteneciente a la tercera generación de este tipo de sistemas, SuperStack II permite alcanzar nuevas metas en lo referente a rendimiento de trabajo en grupo, de acuerdo siempre con las pautas establecidas por la estrategia *Trascend Networking*.

LIBROS



Lotus Notes a su alcance

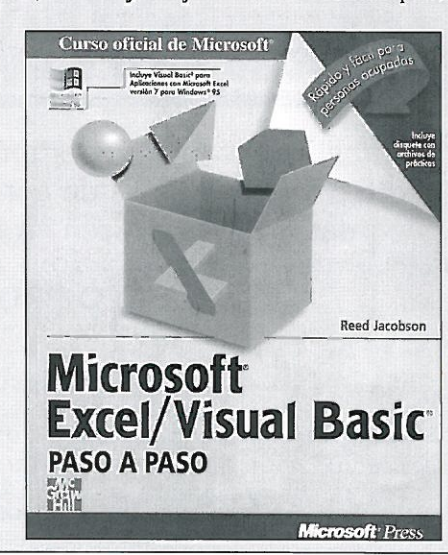
Obra dirigida a los usuarios principiantes de la versión 3 de Lotus Notes, una de las aplicaciones con más éxito dentro de los denominados programas para trabajo en grupo.

Con este libro, el lector podrá conocer a fondo la filosofía de trabajo y el funcionamiento de esta herramienta, implantada en numerosos entornos empresariales de gran prestigio. El libro se divide en tres partes. En una primera parte se exponen los fundamentos básicos que se deben conocer para trabajar con Notes, la instalación y su correcta configuración en Windows. En la segunda parte se explica el entorno y el funcionamiento del correo electrónico y de las bases de datos. Y en la tercera y última parte, el autor se centra en el diseño, la gestión de bases de datos y en el concepto de replicación ,acompañando las explicaciones con numerosos ejemplos.

Editorial: Mc Graw Hill
Autor: Javier Lozano Moreno
368 páginas
Idioma: Castellano
Precio: consultar

Microsoft Excel/Visual Basic paso a paso

Es un libro que ayudará a dominar rápida y eficazmente Visual Basic para aplicaciones, el entorno de desarrollo flexible y ampliable de Microsoft Excel, que facilitará la automatización de sus tareas. Las lecciones, sencillas de utilizar, incluyen objetivos claros y ejemplos reales para que aprenda exactamente lo que necesita a su propio ritmo. Básicamente, con este libro se aprenderá a formatear datos, recorrer la hoja de cálculo, editar y mejorar herramientas para facilitar aún más su uso, utilizar macros, crear informes de una base de datos, crear utilidades de extracción de datos, comunicarse con otras aplicaciones directamente desde Excel utilizando Visual Basic, aplicar sus conocimientos de Visual Basic a otras aplicaciones de Microsoft, etc.



En resumen, es un libro orientado sobre todo a principiantes, pero que bien puede servir también como libro de referencia.

Editorial: Mc Graw Hill
Autor: Reed Jacobson
330 páginas
Idioma: Castellano
Precio: consultar

¡SORTEO DE 50 CONEXIONES A INTERNET!

RELLENA ESTA ENCUESTA Y PARTICIPA EN EL SORTEO

SECCIONES	VALÓRALAS DE 1 A 10
NOTICIAS Y LIBROS	1 2 3 4 5 6 7 8 9 10
CARTAS DEL ILUSO	1 2 3 4 5 6 7 8 9 10
TECNOLOGÍA WEB	1 2 3 4 5 6 7 8 9 10
CURSO DE PROGRAMACIÓN	1 2 3 4 5 6 7 8 9 10
SISTEMAS ABIERTOS	1 2 3 4 5 6 7 8 9 10
GRANDES SISTEMAS	1 2 3 4 5 6 7 8 9 10
CURSO DE REDES	1 2 3 4 5 6 7 8 9 10
SIST. OPERATIVO LINUX	1 2 3 4 5 6 7 8 9 10
CÓMO PROG. UNA DEMO	1 2 3 4 5 6 7 8 9 10
MODOS X DE LA VGA	1 2 3 4 5 6 7 8 9 10
PC INTERNO	1 2 3 4 5 6 7 8 9 10
CURSO VISUAL BASIC	1 2 3 4 5 6 7 8 9 10
CURSO VISUAL C++	1 2 3 4 5 6 7 8 9 10

¿Qué te parece el nivel de los artículos?

- [ ] Muy alto
- [ ] Alto
- [ ] Normal
- [ ] Bajo
- [ ] ....

¿Qué te parece en general el contenido de la revista?

- [ ] Muy bueno
- [ ] Bueno
- [ ] Regular
- [ ] Malo
- [ ] ....

¿Qué cambios te gustaría realizar? ¿Cúales?

- [ ] AÑADIR NUEVAS SECCIONES ________ ________ ________
- [ ] QUITAR SECCIONES ________ ________ ________
- [ ] AMPLIAR LAS ACTUALES ________ ________ ________

¿Te conectas a algún servicio On-Line?

- [ ] Sí
- [ ] No
- [ ] Habitualmente
- [ ] Esporádicamente

¿Cúal?

- [ ] INTERNET
- [ ] INFOVÍA
- [ ] COMPUSERVE
- [ ] BBS
- [ ] OTROS

DATOS PERSONALES

Nombre y Apellidos ____________________

Domicilio ____________________

Población ____________________

C.P. ____________________

Provincia ____________________

Teléfono ____________________

Edad ____________________

Estudios ____________________

Profesión ____________________

Ordenador de que dispone ____________________

Configuración de tu equipo:

Procesador	386	486	PENTIUM < 120	PENTIUM > 120	
Tarjeta Gráfica	VGA	SVGA	65K COLORES	16,7M COLORES	
Tarjeta de Sonido	SOUND BLASTER	SOUND BLASTER 16	GRAVIS ULTRA		
CD-ROM	1X	2X	4X	6X	8X
Módem	2.400	9.600	14.400	28.800	

RELLENA EL CUESTIONARIO Y ENVÍALO A:

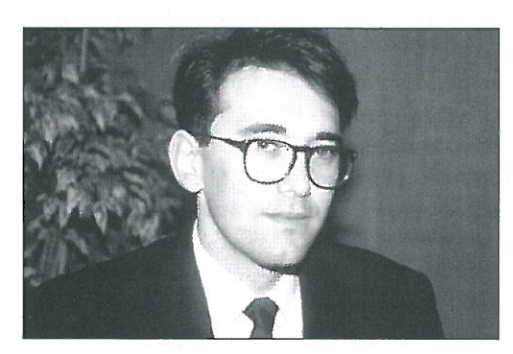
SÓLO PROGRAMADORES (Tower Communications)
Sorteo Internet
C/ Aragoneses, 7 - 28100 Pol. Ind. Alcobendas (Madrid)

Tu revista favorita y la empresa Datagrama te ofrecen la posibilidad de participar en un sorteo de 50 cuentas de conexión a Internet. Para participar, sólo tienes que rellenar los datos que aparecen en esta página y enviarla (sirve fotocopia) a la dirección que aparece unas líneas más arriba.
Nota: Las conexiones son efectivas durante un mes sin limitación diaria.

ACTIVEX: LA EVOLUCIÓN HACIA LOS CONTENIDOS ACTIVOS EN INTERNET (y II)

La tecnología ActiveX es el resultado de un amplio proceso de investigación, realizado conjuntamente con más de 200 fabricantes de software independientes, diseñadores de web y fabricantes de hardware desde septiembre de 1995. En este sentido, desde Microsoft hemos trabajado conjuntamente con compañías como Macromedia para llevar las tecnologías ActiveX a la plataforma Macintosh, aprovechando la experiencia de Macromedia en diversas plataformas. Esta iniciativa muestra nuestra intención de trabajar con otras empresas, en el objetivo final de ofrecer soporte multiplataforma para todas las tecnologías fundamentales de Internet.

Asimismo, hemos elaborado junto con Ncompass Labs un complemento para Netscape Navigator, que permite ver contenido activo dentro de los exploradores de Netscape. El complemento ActiveX permite a los desarrolladores trabajar para los distintos fabricantes, cuando realicen contenido para Internet con las tecnologías ActiveX. En definitiva, pretendemos que ActiveX sea una solución multiplataforma y abierta, que pueda trabajar con todos los estándares del mercado.



Además, hemos anunciado la firma de un acuerdo con Sun Microsystems, para incluir la tecnología Java en los productos de la compañía. Como parte de este acuerdo, desarrollaremos e incluiremos tecnología de desarrollo para Java en la plataforma Windows, incluidos los sistemas operativos Windows 95 y Windows NT.

Hemos anunciado también algunas de estas tecnologías compatibles con Java, englobadas en el compilador Visual J++, cuyo nombre en código era Jacarta. Estas tecnologías incluyen soporte de Java dentro del explorador de web, Microsoft Internet Explorer 3.0; un compilador de alto rendimiento en tiempo real, una herramienta de desarrollo integrada con Java y la unión del lenguaje Java con los objetos estándares COM (Component Object Model) a través de la tecnología Microsoft ActiveX para Internet y PC. De este modo, las librerías de tipo Java serán expuestas como controles ActiveX, permitiendo a las aplicaciones basadas en OLE/COM interoperar y trabajar con los applets de Java.

Asimismo, hemos presentado recientemente la versión beta de Internet Explorer 3.0, que incluye soporte para las actuales características de Netscape Navigator, como Frames, Java y JavaScript, así como soporte para las tecnologías ActiveX, incluyendo los controles ActiveX, ActiveX Documents Objects (documentos activos), el ActiveMovie API, Active VRML y Visual Basic Script.

Otro de nuestros objetivos se basa en ayudar a los Webmasters y a los desarrolladores a utilizar las tecnologías ActiveX, por ello hemos presentado el ActiveX Development Kit, que incluye IIS (Internet Information Server), ficheros de ayuda y el SDK de ActiveX (Software Development Kit). El ActiveX Development Kit también incluye una aplicación completa de entrada de pedidos, diseñada para trabajar con Internet Explorer, el Internet Database Connector, SQL Server e IIS.

Además, hemos creado un estándar para asegurar la integridad y seguridad de los componentes de aplicaciones en Internet a través del uso de una tecnología de firma digital. Esta iniciativa incluye soporte para varios estándares de seguridad, como la certificación X.509 y el estándar PKCS de RSA.

Finalmente, hemos desarrollado, junto con NetManage, el Internet Control Pack, un conjunto de controles ActiveX para Windows, que soportan los protocolos estándar de Internet, permitiendo que esos protocolos sean integrados con las aplicaciones. El Internet Control Pack está formado por los siguientes controles ActiveX: Winsock, FTP, para la transferencia de ficheros; NNTP, que permite a las aplicaciones acceder a las noticias de Usenet y otros servidores de noticias NNTP; SMTP/POP3, que permite a las aplicaciones enviar y recibir correo Internet sin utilizar otras aplicaciones; HTML, que permite a las aplicaciones navegar por documentos HTML; y HTTP.

En definitiva, la creación de un entorno de trabajo de desarrollo, como ActiveX, que integra los estándares de desarrollo corporativos existentes, como OLE, y los estándares emergentes de desarrollo para Internet, como Java, será un factor determinante para que esta tecnología triunfe en el mundo Internet y de las intranets corporativas.

José Antonio Alvarez
Jefe de Producto de Herramientas de Desarrollo de Microsoft Ibérica

LA INTERFAZ CGI: DESCRIPCIÓN AVANZADA

Fernando J. Echevarrieta

En el presente artículo se procederá a la descripción de todos aquellos aspectos de la interfaz CGI que aún no han sido tratados, así como al repaso de los ya vistos, mediante una mayor formalización de los conceptos que, en su momento, se expusieron atendiendo a un criterio didáctico.

Si bien, ante cualquier duda ante la programación CGI, éste pretende, por su completitud, ser el artículo de la serie indispensable de consulta, se aconseja al lector que para el aprendizaje comience por la lectura de los artículos publicados en los números 20 y 21 de Sólo Programadores.

A lo largo del texto se empleará el vocablo *script* del argot para referirse a un programa que cumple con las especificaciónes dictadas por la interfaz CGI. Del mismo modo, al hablar de "*el CGI*" en masulino, se considerará el sustantivo *script* omitido.

FACILITANDO INFORMACIÓN A UN CGI

Supónganse el *script* del listado 1. Como se puede ver, este programa admite una serie de opciones que pasa al programa *ls* de UNIX, el cual muestra el contenido de un directorio. Ejecutándolo desde línea de comando se podrá comprobar cómo, en efecto, al ser ejecutado muestra las pertinentes cabeceras CGI y tras ellas la fuente HTML de una página en la que se ha incluído la salida del comando *ls*. Por ello, algún lector se podría sentir tentado de invocar el script con algo como:
*http://mimaquina.midominio.es/cgi-bin/CGIls -l *.html*

Este lector comprobaría cómo el programa haría caso omiso de las opciones y parámetros que se intentan facilitar.

Esto se debe a que la interfaz CGI utiliza la "línea de comando" para otros fines. Por ello, el envío de información a un *script* CGI se realiza en la mayoría de los casos mediante variables de entorno. De ellas, la más importante de todas es *QUERY_STRING*.

En *QUERY_STRING* queda almacenada cualquier cosa que se se escriba a continuación del primer signo "cierre de interrogación" "?" en el URL correspondiente a un CGI. Es necesario hacer notar que esta información debe ser *URL-codificada* (*URLencoded*), lo que significa que será necesario sustituir los espacios en blanco por el signo "más" (+) y los caracteres especiales deben indicarse en hexadecimal precedidos por un signo "tanto por ciento": *%xx*. El script CGI deberá *URL-descodificar* (*URLdecode*) la cadena.

Pero la interfaz reserva una agradable sorpresa. Siempre que el *script* no esté invocado por un *form* podrá acceder a esta información ya descodificada como si se le hubiera facilitado realmente desde línea de comando. Es decir, como si no se tratara de un CGI. Esto, por ejemplo, significa que en una shell UNIX se podrá acceder a ella mediante las variables *$1, $2 ... $n* y en un programa en C mediante *argv*.

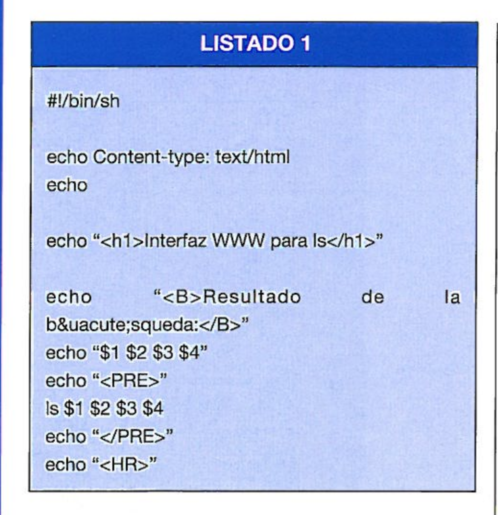
Así pues, continuando con el ejemplo, para facilitar la información de parámetros y opciones a CGIls, bastaría con teclear:
http://mimaquina.midominio.es/cgi-bin/CGIls?-lF+.html*

DOCUMENTOS *ISINDEX*

Una de las etiquetas de HTML que se menciona en todos los lugares pero la



Una vez realizada una primera toma de contacto con los conceptos fundamentales asociados a la programación CGI, y concretados posteriormente en su aplicación al procesamiento de *forms*, ahora se terminarán de exponer las facilidades que proporciona la interfaz.



LISTADO 1

```
#!/bin/sh

echo Content-type: text/html
echo

echo "<h1>Interfaz WWW para ls</h1>"

echo "<B>Resultado de la
b&uacute;squeda:</B>"
echo "$1 $2 $3 $4"
echo "<PRE>"
ls $1 $2 $3 $4
echo "</PRE>"
echo "<HR>"
```

mayoría pasa por alto porque realmente no sabe qué uso tiene, es la etiqueta *<ISINDEX>*, que formalmente declara que un documento es un "índice de búsqueda".

Esta etiqueta debe ser empleada únicamente en la salida de CGIs y genera una página similar a un *form*, con un campo en el que el usuario puede introducir una serie de parámetros. La ventaja es que no hay necesidad de URL-codificar estos parámetros, por lo que se pueden teclear manualmente de la misma forma que se escribirían en la línea de comando de un programa cualquiera. Al pulsar la tecla <ENTER>, el CGI se invocará a sí mismo (de ahí que únicamente tenga sentido en CGIs y no en documentos estáticos), generando una variable *QUERY_STRING* adecuada con los parámetros que el usuario ha introducido pero ya URL-codificados (lo que se puede observar si se tiene la opción mostrar URL o *show location* activada). Ahora bien, como se indicaba en el punto anterior, si el CGI no está invocado por un *form*, recibe los parámetros ya URL-descodifcados como cualquier programa y, en efecto, un documento *ISINDEX*, aunque tiene apariencia de *form*, no lo es.

Por tanto, hasta ahora era necesario uno de los dos pasos: o URL-codificar los parámetros de un CGI para generar un URL que lo invocara, o enviarlos a través de un *form*, lo que implicaba que el CGI en cuestión debía URL-descodificarlos. Mediante el uso de *<ISINDEX>* se hace posible escribir los parámetros de un CGI de la forma "habitual".

Así, por ejemplo, si se invoca el CGI del listado 2, al que podría, por ejemplo llamarse *CGIls*, mostrará un documento como el que aparece en la figura 1. En el recuadro, sí podría esta vez teclearse en el cuadro
*-lF *.html*

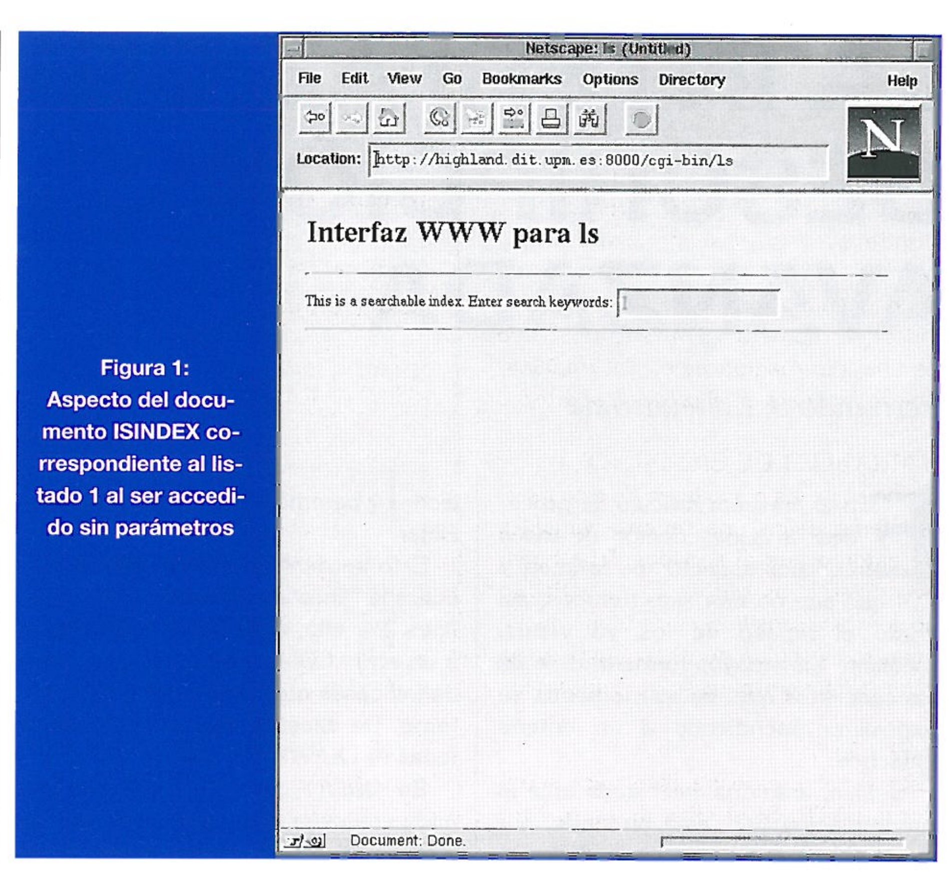


Figura 1: Aspecto del documento ISINDEX correspondiente al listado 1 al ser accedido sin parámetros

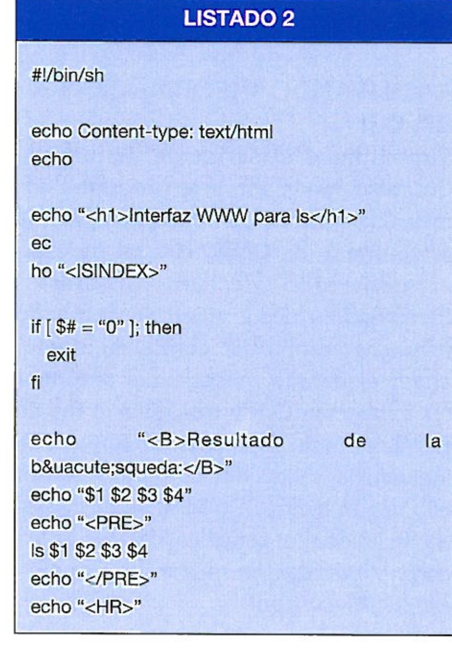
FORMALIZACIÓN DE LA INTERFAZ

Para concluir con la especificación CGI se presentará una formalización parcial de la misma, para lo cual se ha tomado como referencia principal el draft de RFC del 8 de Enero de 1996, cuya última modificación es del 15 Febrero, escrito por D.R.T. Robinson (*http://www.ast.cam.ac.uk/~drtr/ cgi-spec.html*), que contiene las especificaciones de la versión 1.1 de la interfaz CGI. Se ha optado por dejar a un lado todos los formalismos como el uso (salvo en uno o dos casos) de la notacion EBNF, y se han incluído diversos ejemplos "del mundo real" para hacer comprensibles ciertos aspectos para los que un RFC no es la mejor guía didáctica.

En cualquier caso, se recuerda al lector que todas estas especificaciones no han surgido de una autoridad que las ha impuesto sino, como gran parte del trabajo en internet, de las conversaciones mantenidas a través de una lista de correo abierta, la *www-talk*, entre las que han destacado las contribuciones de Rob McCool, John Franks, Ari Luotonen, George Phillips y Tony Sanders.

URL DE UN SCRIPT

Formalizando todo lo expuesto hasta el momento, se puede decir que la forma



LISTADO 2

```
#!/bin/sh

echo Content-type: text/html
echo

echo "<h1>Interfaz WWW para ls</h1>"
ec
ho "<ISINDEX>"

if [ $# = "0" ]; then
  exit
fi

echo "<B>Resultado de la
b&uacute;squeda:</B>"
echo "$1 $2 $3 $4"
echo "<PRE>"
ls $1 $2 $3 $4
echo "</PRE>"
echo "<HR>"
```

Documento ISINDEX

de invocar un *script* es acceder a una URL que lo identifica y se construye de la forma:

script-url=protocolo"://"SERVER_NAME":"SERVER_PORT enc_scrip enc_path_info"?"QUERY_STRING

donde *enc_script* se corresponde al nombre del *script*, bajo el nombre lógico del directorio asignado a CGIs por el servidor (ver número 19) y *enc_path_info* es una información adicional opcional que se explicará en el próximo apartado.

VARIABLES DE ENTORNO

Como se ha visto a lo largo de esta serie, la interfaz define una serie de variables de entorno que el servidor debe hacer accesibles a los *scripts*. Estas variables no son *case sensitive*, es decir, no distinguen mayúsculas de minúsculas, y un valor NULL equivale a su no definición.

AUTH_TYPE

Esta variable es específica para servidores HTTP. Si el URL del *script* requiere autenticación de acceso se emplea esta variable para indicar el método de protección.

CONTENT_LENGTH

Indica el tamaño del la información que acompaña a la invocación (*attachment*). Si no existe ninguna entidad asociada toma el valor NULL. Esta información puede provenir, por ejemplo de un método *POST* o *PUT* de HTTP. Un ejemplo sencillo es la información facilitada por un *form*.

CONTENT_TYPE

Indica el tipo *MIME* correspondiente a la información que acompaña a la invocación. Continuando con el ejemplo del *form*, su valor podría ser: *application/x-www-form-urlencoded*.

Sólo en caso de que esta variable no tenga valor, el *script* debe tratar de deducir el tipo de la información examinándola. Si aún así no se puede determinar de qué tipo es, se asumirá el tipo *application/octect-stream*.

GATEWAY_INTERFACE

Indica la versión de la especificación CGI que cumple el servidor, de la forma "CGI/revisión", por ejemplo "CGI/1.1"

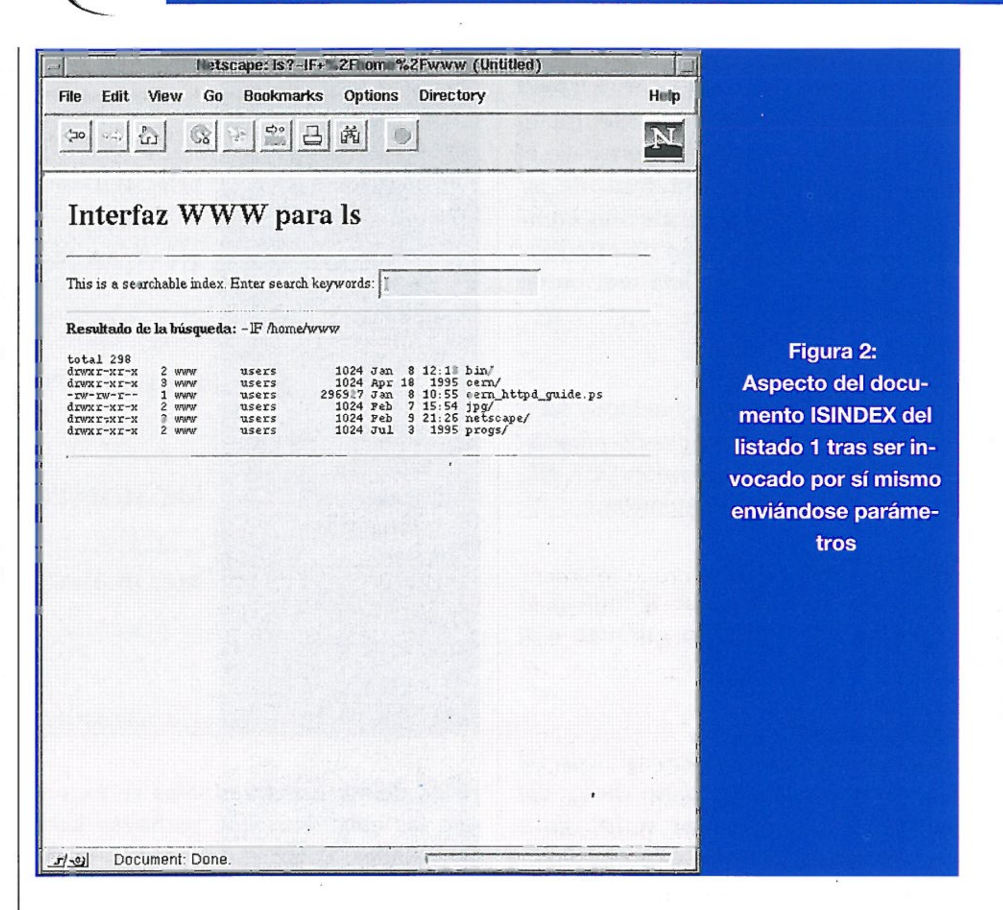


Figura 2: Aspecto del documento ISINDEX del listado 1 tras ser invocado por sí mismo enviándose parámetros

PATH_INFO

La interfaz CGI permite que cierta información adicional vaya incrustada en el URL de invocación al CGI. Esta información se sitúa tras el nombre del CGI en el URL y precediendo, si existe, al signo de interrogación "?" que indicaría el comienzo del valor de *QUERY_STRING*. Es lo que en el URL del *script* se ha denominado *enc_path_info*.

El uso más habitual es indicar localizaciones de ficheros (*paths*) al CGI. Mirando hacia atrás, se puede ahora identificar esta forma de facilitar información a un *script* en el mecanismo de realización de imágenes-mapa sensibles que se presentaron en el número 17, en el que se invocaba al *script* que interpretaba el mapa como en el siguiente ejemplo:

<a href=/cgi-bin/htimage/sp/fotomapa.map>

en el que, claramente, */sp/fotomapa.map* es la información adicional que el intérprete de mapas necesita para localizar el mapa.

PATH_TRANSLATED

Indica el valor de *PATH_INFO* tras el mapeado del servidor de direcciones lógicas a direcciones físicas. La interfaz CGI no obliga a que exista valor para esta variable por motivos ya sea de seguridad o de otra clase, como el hecho de que un mapeo puede no conducir a un objeto físico, como por ejemplo, un fichero.

De esta variable depende la implementación, por lo que si se desea hacer portables los scripts CGI conviene no utilizarla.

QUERY_STRING

Incluye toda la información que sigue al signo interrogación "?" en la invocación al CGI, como se ha explicado en el primer punto del artículo.

REMOTE_ADDR

Indica la dirección IP del agente que invocó al CGI. Esta no tiene por qué ser necesariamente la del cliente ya que si, por ejemplo, éste accede al servidor a través de otro servidor *proxy*, tomará el valor de la dirección IP de este último.

REMOTE_HOST

Se comporta de forma análoga a la variable *REMOTE_ADDR*, indicando en este caso el nombre completo (o NULL) del agente que invocó al CGI.

REMOTE_IDENT
Si el servidor remoto admite *ident* (descrito en el RFC931) facilita el nombre del usuario al que pertenece el cliente. Este valor debe únicamente ser considerado como información complementaria o elaboración de estadísticas, pero nunca como información de autenticación.

REQUEST_METHOD
Esta variable sí es *case sensitive* y facilita el método HTTP empleado para facilitar información al servidor: *GET*, *HEAD*, *POST* o cualquier extensión.

SCRIPT_NAME
Proporciona el URL que identifica al *script* CGI. Es útil para referirse a sí mismo.

SERVER_NAME
Proporciona el nombre del servidor, tal y como se indicó en la parte <host> del URL de invocación del *script*. Este nombre puede, por tanto ser una dirección IP, un alias de DNS o un nombre propiamente dicho.

SERVER_PORT
Funciona análogamente a *SERVER_NAME*, indicando el puerto de atención del servidor.

SERVER_PROTOCOL
Facilita la versión y extensión del protocolo empleado por el servidor de información. Por ejemplo:
SERVER_PROTOCOL=HTTP-version|extension-version

SERVER_SOFTWARE
Proporciona el nombre y versión del software servidor que sirve la petición o ejecuta la pasarela.

VARIABLES *HTTP_**
Estas variables se definen únicamente para peticiones realizadas mediante HTTP y su interpretación es función del valor de *SERVER_PROTOCOL*. Se suele definir una variable HTTP_ para cada una de las cabeceras que se reciben del cliente (si se reciben), que toma el nombre de HTTP_(nombre_de_cabecera). Al darse valor a las variables se realiza un cambio del carácter guión (-) por el carácter subrayado (_). A la hora de definir estas variables se excluyen las cabeceras que ya hayan sido procesadas, como *Authorization*, o cuya información figure en otras variables de entorno, como *Content-type* o *Content-length*.

Entre las variables HTTP_* más usadas se encuentran:

HTTP_ACCEPT
Indica los tipos MIME aceptados por el servidor separados por comas "," en la forma habitual: tipo/subtipo, tipo/subtipo ...

HTTP_USER_AGENT
Indica el *browser* empleado por el cliente de la forma software/versión biblioteca/versión.

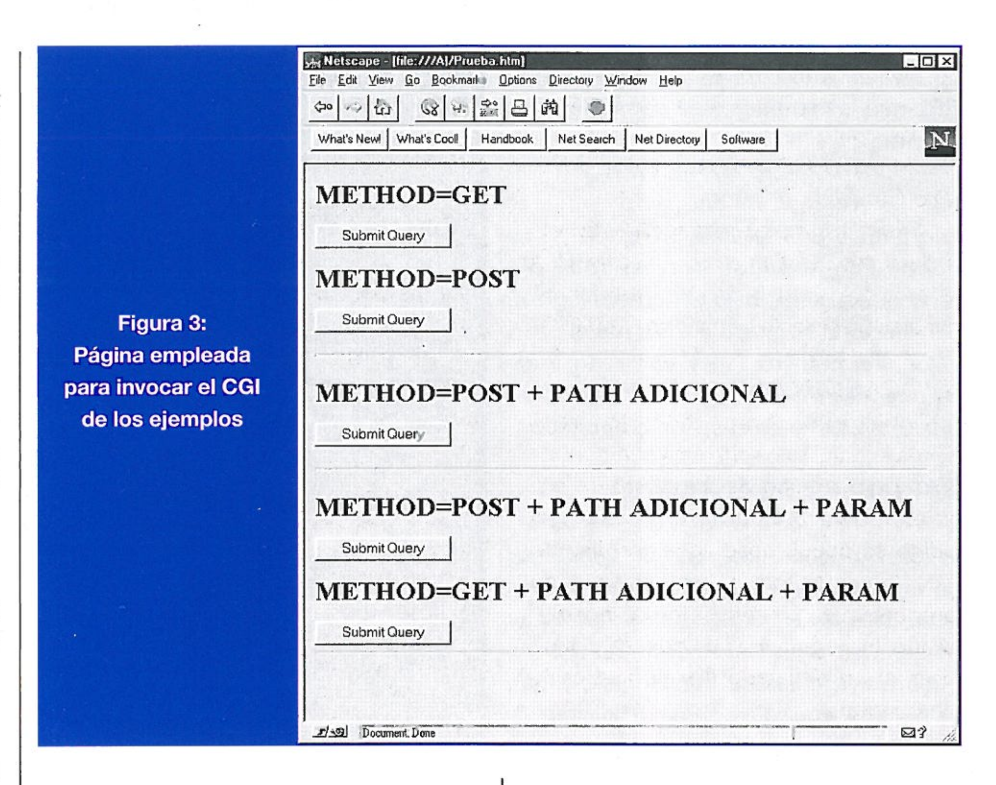


Figura 3: Página empleada para invocar el CGI de los ejemplos

SALIDA DE DATOS DEL CGI
La interfaz especifica que un *script* CGI siempre debe devolver algo, y esta salida debe realizarse a través de la salida estándar del proceso. En función de cómo se realiza esta salida, se clasifican los *scripts* CGI en dos tipos diferentes:

Salida NPH
Este tipo de *scripts* debe devolver un mensaje HTTP completo. El servidor enviará este mensaje completo al cliente sin realizar ningún tipo de análisis de las cabeceras (*No Parse Headers*) y el *buffering* intermedio debe reducirse a la mínima expresión. Este tipo de *scripts* será muy importante, ya que en él se basa la realización de los documentos dinámicos basados en la técnica del *server-push* propuesta por Netscape.

Salida PH
Es el tipo más habitual de *script*. Debe devolver una salida del tipo:
**(Cabecera_CGI|Cabecera_HTTP)NL[Cuerpo_Entidad]*
sintáxis expresada en *EBNF* que significa repetición de una o más veces la secuencia de una cabecera CGI o una cabecera HTTP seguidas de un avance de línea y el cuerpo de la información.

La especificación indica que el servidor deberá realizar una traducción de las cabeceras HTTP para adaptarlas del formato de la máquina que ejecuta el *script* al empleado por el protocolo HTTP. Por ejemplo, el carácter NL generado por un script UNIX será automáticamente traducido a los caracteres CR+LF empleados por el protocolo.

La especificación de la intefaz exige que se proporcione al menos una cabecera CGI, que podrá pertencer a uno de los siguientes tipos:

Content-type: Define el tipo *MIME* a que corresponde la información que proporciona el *script*. Esta cabecera es de uso obligatorio si la respuesta del CGI incluye un cuerpo de información.

Location: Actua como un puntero devolviendo un URL en lugar de la información. Sólo se debe emplear si *REQUEST_METHOD* toma los valores *HEAD* o *GET*.

Status: Se emplea para facilitar un valor de status definido en la especificación del protocolo HTTP. En caso de que no se indique, el cliente tomará un *"200 OK"*.

INVOCACIÓN DE UN CGI

Con objeto de aclarar todos los conceptos expuestos sobre invocación de CGIs y generación de variables de entorno, se invocará un mismo CGI, cuyo código se puede observar en el listado 3, de diferentes formas para estudiar su comportamiento. Para ello, se ha elaborado la página HTML que se puede observar en el listado 4 y corresponde a la figura 3.

LISTADO 3

```
#!/bin/sh

echo HTTP/1.0 200 OK
echo Content-type: text/plain
echo Server: $SERVER_SOFTWARE
echo

echo CGI/1.0 test script report:
echo

echo argc is $#. argv is "$*".
echo

echo SERVER_SOFTWARE = $SERVER_SOFTWA-
RE
echo SERVER_NAME = $SERVER_NAME
echo GATEWAY_INTERFACE = $GATEWAY_IN-
TERFACE
echo SERVER_PROTOCOL = $SERVER_PROTO-
COL
echo SERVER_PORT = $SERVER_PORT
echo REQUEST_METHOD = $REQUEST_METHOD
echo HTTP_ACCEPT = "$HTTP_ACCEPT"
echo PATH_INFO = $PATH_INFO
echo PATH_TRANSLATED = $PATH_TRANSLATED
echo SCRIPT_NAME = $SCRIPT_NAME
echo QUERY_STRING = $QUERY_STRING
echo REMOTE_HOST = $REMOTE_HOST
echo REMOTE_ADDR = $REMOTE_ADDR
echo REMOTE_USER = $REMOTE_USER
echo CONTENT_TYPE = $CONTENT_TYPE
echo CONTENT_LENGTH = $CONTENT_LENGTH
echo AUTH_TYPE=$AUTH_TYPE
echo HTTP_REFERER=$HTTP_REFERER
echo HTTP_USER_AGENT=$HTTP_USER_AGENT
echo REFERER_URL=$REFERER_URL
echo REMOTE_IDENT=$REMOTE_IDENT
```

Código del CGI invocado en los ejemplos

LISTADO 4

```
<H1>METHOD=GET</H1>
<FORM ACTION=http://highland.dit.upm.es:8000/cgi-bin/nph-test-cgi METHOD=GET>
<INPUT TYPE=HIDDEN VALUE="SOLO PROGRAMADORES" NAME=NOMBRE>
<INPUT TYPE=SUBMIT>
</FORM>

<H1>METHOD=POST</H1>
<FORM ACTION=http://highland.dit.upm.es:8000/cgi-bin/nph-test-cgi METHOD=POST>
<INPUT TYPE=HIDDEN VALUE="SOLO PROGRAMADORES" NAME=NOMBRE>
<INPUT TYPE=SUBMIT>
</FORM>

<H1>METHOD=POST + PATH ADICIONAL</H1>
<FORM ACTION=http://highland.dit.upm.es:8000/cgi-bin/nph-test-cgi/path/adicional METHOD=POST>
<INPUT TYPE=HIDDEN VALUE="SOLO PROGRAMADORES" NAME=NOMBRE>
<INPUT TYPE=SUBMIT>
</FORM>

<H1>METHOD=POST + PATH ADICIONAL + PARAM</H1>
<FORM ACTION=http://highland.dit.upm.es:8000/cgi-bin/nph-test-cgi/path/adicional?param1 MET-
HOD=POST>
<INPUT TYPE=HIDDEN VALUE="SOLO PROGRAMADORES" NAME=NOMBRE>
<INPUT TYPE=SUBMIT>

<H1>METHOD=GET + PATH ADICIONAL + PARAM</H1>
<FORM ACTION=http://highland.dit.upm.es:8000/cgi-bin/nph-test-cgi/path/adicional?param1 MET-
HOD=GET>
<INPUT TYPE=HIDDEN VALUE="SOLO PROGRAMADORES" NAME=NOMBRE>
<INPUT TYPE=SUBMIT>
```

Fuente HTML de las pàgina empleada para realizar las invocaciones de los ejemplos

En el listado 5 se puede observar el resultado de la primera invocación al CGI, que se puede emplear como ejemplo práctico para comprender mejor el significado de las variables:

<FORM ACTION=http://highland.dit.upm.es:8000/cgi-bin/nph-test-cgi METHOD=GET>

Aquí, la variable *SERVER_SOFTWARE* indica que el servidor de información es el servidor de WWW del CERN versión 3.0, precisamente la versión que se facilitó en el número 19 de Sólo Programadores. *SERVER_NAME* informa de que se trata de *highland*, el servidor del autor. Asimismo, este servidor se ajusta a la especificación CGI 1.1, que es la expuesta en este mismo artículo, como anuncia la variable *GATEWAY_INTERFACE*. El protocolo de acceso y su versión son HTTP 1.0 (*SERVER_PROTOCOL*) y se puede ver que el servicio atiende en un puerto no estándar, el 8000 (*SERVER_PORT*). El método empleado para el envío de los datos del *form*, que como se puede comprobar en el listado 4, se trata únicamente de un campo oculto con la cadena "SÓLO PROGRAMADORES", es *GET* (si no fuera así habría que preocuparse). Y *HTTP_ACCEPT* informa de los tipos *MIME* aceptados.

La invocación se realizó sin información adicional. Por lo tanto, las variables *PATH_INFO* y *PATH_TRANSLATED* deben aparecer vacías o, de ser leidas desde un programa C, deberían ser NULL. *SCRIPT_NAME* facilita el nombre del *script* y, como se puede observar, no corresponde ni a un URL completo ni a un nombre físico (real) sino al URL relativo al URL base del servidor. Y como era de esperar, en el punto más importante de estos ejemplos, el empleo de un método *GET* implica el envío de los datos del *form* a través de la variable *QUERY_STRING*, en la adecuada secuencia de cadenas nombre=variable separadas por *ampersands* y URL-codificadas (obsérvese el signo "más" entre las palabras).

La variable *REMOTE_HOST* indica que la conexión se realizó desde cuco en el DIT, pero no existe forma de saber si fue realizada directamente por un cliente o por un servidor *proxy*. Lo que

LISTADO 5

```
CGI/1.0 test script report:

argc is 0. argv is .

SERVER_SOFTWARE = CERN/3.0
SERVER_NAME = highland.dit.upm.es
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 8000
REQUEST_METHOD = GET
HTTP_ACCEPT = image/gif, image/x-xbitmap,
image/jpeg, image/pjpeg, */*
PATH_INFO =
PATH_TRANSLATED =
SCRIPT_NAME = /cgi-bin/nph-test-cgi
QUERY_STRING = NOMBRE=SOLO+PROGRA-
MADORES
REMOTE_HOST = cuco.dit.upm.es
REMOTE_ADDR = 138.4.2.11
REMOTE_USER =
CONTENT_TYPE =
CONTENT_LENGTH =
AUTH_TYPE=
HTTP_REFERER=http://highland.dit.upm.es:800
0/Prueba.htm
HTTP_USER_AGENT=Mozilla/2.0 (X11; I; SunOS
4.1.3 sun4)
REFERER_URL=http://highland.dit.upm.es:8000
/Prueba.htm
REMOTE_IDENT=
```

Invocación mediante el método GET

LISTADO 6

```
CGI/1.0 test script report:

argc is 0. argv is .

SERVER_SOFTWARE = CERN/3.0
SERVER_NAME = highland.dit.upm.es
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 8000
REQUEST_METHOD = POST
HTTP_ACCEPT = image/gif, image/x-xbitmap,
image/jpeg, image/pjpeg, */*
PATH_INFO =
PATH_TRANSLATED =
SCRIPT_NAME = /cgi-bin/nph-test-cgi
QUERY_STRING =
REMOTE_HOST = cuco.dit.upm.es
REMOTE_ADDR = 138.4.2.11
REMOTE_USER =
CONTENT_TYPE = application/x-www-form-ur-
lencoded
CONTENT_LENGTH = 25
AUTH_TYPE=
HTTP_REFERER=http://highland.dit.upm.es:800
0/Prueba.htm
HTTP_USER_AGENT=Mozilla/2.0 (X11; I; SunOS
4.1.3 sun4)
REFERER_URL=http://highland.dit.upm.es:8000
/Prueba.htm
REMOTE_IDENT=
```

Invocación mediante el método POST

LISTADO 7

```
CGI/1.0 test script report:

argc is 0. argv is .

SERVER_SOFTWARE = CERN/3.0
SERVER_NAME = highland.dit.upm.es
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 8000
REQUEST_METHOD = POST
HTTP_ACCEPT = image/gif, image/x-xbitmap,
image/jpeg, image/pjpeg, */*
PATH_INFO = /path/adicional
PATH_TRANSLATED =
/hypertext/WWW/path/adicional
SCRIPT_NAME = /cgi-bin/nph-test-cgi
QUERY_STRING =
REMOTE_HOST = cuco.dit.upm.es
REMOTE_ADDR = 138.4.2.11
REMOTE_USER =
CONTENT_TYPE = application/x-www-form-ur-
lencoded
CONTENT_LENGTH = 25
AUTH_TYPE=
HTTP_REFERER=http://highland.dit.upm.es:800
0/Prueba.htm
HTTP_USER_AGENT=Mozilla/2.0 (X11; I; SunOS
4.1.3 sun4)
REFERER_URL=http://highland.dit.upm.es:8000
/Prueba.htm
REMOTE_IDENT=
```

Invocación mediante el método POST incorporando información adicional

sí se facilita en *REMOTE_ADDR* es su dirección IP. *REMOTE_USER*, como en el 90% de los casos aparece vacía, ya que cuco no tiene instalado el demonio adecuado para facilitar el nombre del usuario. *CONTENT_TYPE* y *CONTENT_LENGTH* aparecen también vacías, ya que no se ha proporcionado ningún *attachment* al emplear un método *GET*, que facilitó la información en la variable *QUERY_STRING*. Y *AUTH_TYPE* aparece también vacía, ya que el CGI invocado no era un documento protegido.

Posteriormente, aparecen algunas variables *HTTP_*, *HTTP_USER_AGENT* indica que el usuario se conectó desde una máquina Sun 4 ejecutando el sistema operativo SunOs 4.1.3 y empleando como *browser* el Netscape 2.0 (Netscape se identifica mediante la cadena Mozilla) para X11 (*X Window*). Esta es otra de las variables más útiles, ya que en función del *browser* empleado se puede enviar una cosa u otra. En este caso, se sabe que la respuesta del CGI puede contener *frames*, tablas y *applets* Java. Si se tratara de otro *browser* o de otro Sistema Operativo quizá únicamente se podría mostrar una tabla con texto <PRE> procesado.

Por último, *HTTP_REFERER* y *REFERER_URL* proporciona el URL del documento desde el cual se accedió al *script*, lo cual es un interesante método de conocer qué lugares del WWW apuntan a nuestro servidor o *Home Page*. Por ejemplo, basta colocar un CGI como *Home Page* que vaya escribiendo en un fichero el valor de esta variable para saber en qué páginas de la red figuramos y cómo llegaron a saber de nosotros.

DISTINTAS FORMAS DE INVOCAR EL MISMO CGI

En el ejemplo del listado 6 se ha empleado exactamente el mismo CGI y el mismo código HTML, salvo que se ha sustituido el método *GET* por el método *POST*.

<FORM ACTION=http://highland.dit.upm.es:8000/cgi-bin/nph-test-cgi METHOD=POST>

Como se puede apreciar, la mayor parte del entorno del CGI no cambia salvo las variables *HTTP_METHOD*, que ha pasado a valer *POST*, y que ahora existe un valor para *CONTENT_TYPE*, *application/x-www-form-urlencoded*, que indica que se le ha enviado un *attachment* con información procedente de un *form* HTML. También existe un valor para *CONTENT_LENGTH*, 25, que es la longitud de la cadena "NOMBRE=SOLO+PROGRAMADORES". Sin embargo, la información no aparece por ningún sitio y la variable *QUERY_STRING* se encuentra vacía. Esto es debido simplemente al propio método *POST* que, como indica la interfaz, envía la información a la entrada estándar del CGI. Como el CGI de ejemplo no lee la entrada estándar, simplemente, no recibe la información.

En el caso del listado 7, la invocación se realiza mediante el método *POST* incorporando información adicional.

<FORM ACTION=http://highland.dit.upm.es:8000/cgi-bin/nph-test-cgi/path/ adicional METHOD=POST>

En este caso, la respuesta es prácticamente idéntica al ejemplo del listado 4, salvo porque este *path* adicional se

LISTADO 8

```
CGI/1.0 test script report:

argc is 1. argv is param1.

SERVER_SOFTWARE = CERN/3.0
SERVER_NAME = highland.dit.upm.es
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 8000
REQUEST_METHOD = POST
HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
PATH_INFO = /path/adicional
PATH_TRANSLATED = /hypertext/WWW/path/adicional
SCRIPT_NAME = /cgi-bin/nph-test-cgi
QUERY_STRING = param1
REMOTE_HOST = nexus.dit.upm.es
REMOTE_ADDR = 138.4.22.26
REMOTE_USER =
CONTENT_TYPE = application/x-www-form-urlencoded
CONTENT_LENGTH = 51
AUTH_TYPE=
HTTP_REFERER=file:///Al/Prueba.htm
HTTP_USER_AGENT=Mozilla/2.01 (Win95; I)
REFERER_URL=file:///Al/Prueba.htm
REMOTE_IDENT=
```

Invocación mediante el método GET incorporando información adicional y parámetros

refleja ahora en las variables *PATH_INFO*, que proporciona exactamente la información facilitada */path/adicional,* y *PATH_TRANSLATED,* que proporciona el nombre físico correspondiente tras realizar el servidor el mapeo de direcciones lógicas (URLs) a físicas */hypertext/WWW/path/adicional.* Con este ejemplo ya puede el lector realizar su propio CGI de tratamiento de imágenes mapa. No hay que olvidar que una URL podría no corresponderse con una dirección física, como se mencionaba anteriormente en esta misma entrega.

EJEMPLO DE LO QUE NO HAY QUE HACER

En el ejemplo del listado 8 se ha "rizado el rizo", al emplear un método *POST* con información adicional y, además, incorporando parámetros al CGI:

<FORM ACTION=http://highland.dit.upm.es:8000/cgi-bin/nph-test-cgi/path/ adicional METHOD=POST>

En el resultado se puede ver cómo el valor de la variable *CONTENT_LENGTH* ha dejado de ser fiable y, de hecho, si se analizara la entrada en este ejemplo,se vería que vale:

LISTADO 9

```
CGI/1.0 test script report:

argc is 1. argv is param1.

SERVER_SOFTWARE = CERN/3.0
SERVER_NAME = highland.dit.upm.es
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 8000
REQUEST_METHOD = POST
HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
PATH_INFO = /path/adicional
PATH_TRANSLATED = /hypertext/WWW/path/adicional
SCRIPT_NAME = /cgi-bin/nph-test-cgi
QUERY_STRING = param1
REMOTE_HOST = nexus.dit.upm.es
REMOTE_ADDR = 138.4.22.26
REMOTE_USER =
CONTENT_TYPE = application/x-www-form-urlencoded
CONTENT_LENGTH = 51
AUTH_TYPE=
HTTP_REFERER=file:///Al/Prueba.htm
HTTP_USER_AGENT=Mozilla/2.01 (Win95; I)
REFERER_URL=file:///Al/Prueba.htm
REMOTE_IDENT=
```

Invocación mediante el método GET incorporando información adicional y parámetros

NOMBRE=SOLO+PROGRAMADORES&NOMBRE=SOLO+PROGRAMADORES

lo cual es incorrecto. Si bien haría recuperable la información, en este caso, el comportamiento no está definido.

En el caso del listado 9, el ejemplo es análogo empleando el método *GET.* En este caso el problema va más allá, ya que existe un conflicto debido a que, al incorporarse parámetros a la invocación del CGI se transmite su valor a través de la variable *QUERY_STRING,* que es precisamete la que debería transmitir la información recogida por el *form.*

<FORM ACTION=http://highland.dit.upm.es:8000/cgi-bin/nph-test-cgi/path/ adicional?param1 METHOD=GET>

En cualquier caso, tanto la información sobre *PATH_ADICIONAL* como los parámetros han llegado a destino en los dos casos. Así pues, no se debe tratar de emplear los tres mecanismos: *form,* información adicional y parámetros a la vez. Puede que un caso concreto funcione, pero el comportamiento será impredecible en otros servidores.

En estos ejemplos, se puede ver que ha cambiado también el valor de otras variables, ya que esta vez la invocación se hizo desde un fichero local en un sistema Windows 95.

RECOMENDACIONES PARA EL DESARROLLO DE SCRIPTS

A la hora de realizar *scripts,* por seguridad o por estilo, conviene seguir una de las siguientes recomendaciones:

1. Rechazar aquellos métodos que no se esperan, como por ejemplo *DELETE,* mostrando una cabecera de Status con el valor
405 Method Not Allowed

2. Si no se usa *PATH_INFO* y se le indica con un valor distinto de *NULL,* es recomendable contestar con una cabecera de Status con el valor
404 Not Found

3. Cuando el CGI está procesando un *form,* el valor de *CONTENT_TYPE* debe ser "*application/x-www-form-urlencoded*" o "*multipart/form-data*". En caso contrario, se debe contestar con una cabecera de Status con el valor
400

4. Por razones de seguridad es importante revisar el valor de las variables *PATH_INFO, PATH_TRANSLATED* y *SCRIPT_NAME,* anular de las mismas la doble barra (//) y si en las mismas se encuentra la referencia a directorio anterior (..) o, incluso al directorio actual (.) no usarlas jamás en una llamada al sistema. De hecho, en estos casos suele ser aconsejable suspender la ejecución del *script* informando de la causa mediante una cabecera de Status:
404 Not Found.

Esto se debe al avieso comportamiento de algunos internautas, que aprovechan ciertas entradas sin protejer como éstas para realizar acciones sobre directorios del sistema a los que no deberían tener acceso ya que si, por ejemplo, se permite que un CGI acceda a un directorio anterior al suyo, puede trascender los límites de seguridad del servidor de información.

5. No se deben retornar documentos con contenido *text/html* con URLs relativas y que no incluyan la etiqueta *<BASE>.*

6. Las cabeceras CGI deben figurar antes que las cabeceras HTTP.

EVALUACIÓN DE EXPRESIONES

José C. Remiro



En el anterior artículo se presentaron un conjunto de unidades que contenían funciones y procedimientos básicos para la utilización de listas lineales y árboles binarios, cuyos nodos podían contener tanto caracteres como números. El fin de éstos era ayudar a realizar el análisis sintáctico y la evaluación de expresiones cuya sintaxis estaba definida mediante una serie de grafos. En esta entrega se describirán los diferentes pasos para realizar estos procesos.

ANÁLISIS SINTÁCTICO

La utilización de un grafo para describir la sintáxis de una expresión facilita la programación de un conjunto de procedimientos para decidir si una expresión es sintácticamente correcta.

Básicamente, los grafos que representan a cada uno de los elementos de una expresión siguen el esquema mostrado en el cuadro 1.

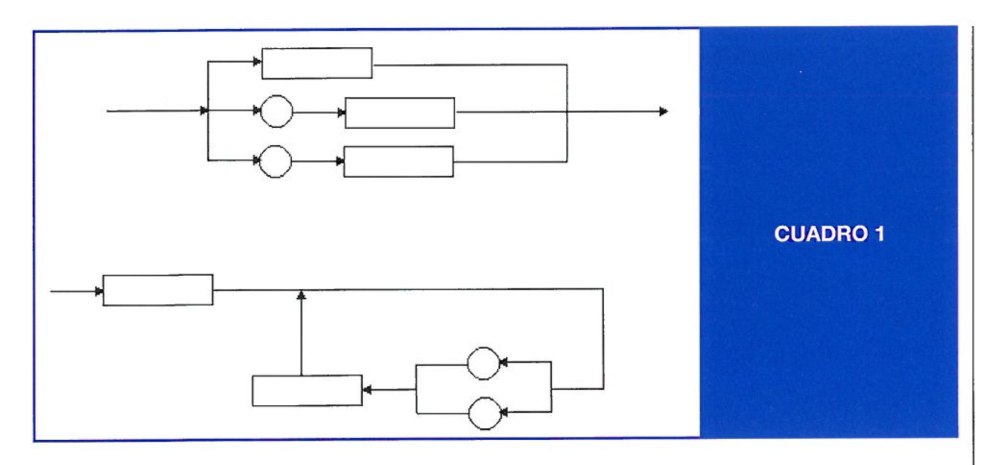
En el primer grafo del cuadro 1 se puede observar que hay una serie de opciones, entre las que hay que decidir si se encuentra un determinado símbolo terminal (representados por círculos), a los que sigue un símbolo no terminal (representado por un rectángulo) o bien hay un símbolo no terminal. Este tipo de grafo se representará en el programa como una serie de instrucciones *if* anidadas que comprobarán si dichos elementos terminales se encuentran presentes en la parte inicial de la expresión que se está analizando. Si es así, se procederá a obtener el siguiente símbolo significativo de la expresión y se procederá a llamar al procedimiento que analiza el símbolo no terminal. Si directamente el símbolo es no terminal, se procederá a realizar la llamada a un procedimiento que representa a dicho símbolo. Es importante que el primer elemento terminal de un símbolo no terminal, en el tipo de grafo que se está comentando actualmente, sea distinto de los diferentes símbolos no terminales presentes en el grafo, pues de lo contrario no se podría decidir a qué procedimiento hay que llamar ni qué camino se sigue en el grafo.

En el segundo grafo del cuadro 2 se puede observar que en primer lugar habrá un símbolo no terminal. Siguiendo a éste, puede o no haber un símbolo terminal seguido de un símbolo no terminal, o bien finalice el grafo. Esta situación se programará llamando inicialmente al procedimiento que reconoce el símbolo no terminal y, cuando éste termine, se procederá a comprobar la existencia de dichos símbolos no terminales procediendo, si es preciso, a la llamada del procedimiento que representa al símbolo no terminal. Esta llamada deberá repetirse mientras se encuentre con los símbolos terminales una vez que el procedimiento asociado al símbolo no terminal finaliza el reconocimiento de la subexpresión. La implementación de esta situación se realizará mediante una instrucción *while*.

En el cuadro 2 se muestra el esqueleto de los diferentes procedimientos que se corresponden con los dos grafos que se describieron anteriormente.

El procedimiento con el que se inicia el análisis sintáctico es *expresión_simple*. Este procedimiento acepta una cadena de caracteres y una variable que apunta al símbolo de la cadena que se está tratando actualmente, e intenta comprobar si ésta contiene una expre-

Los árboles son una buena estructura de datos para representar y evaluar expresiones. Una vez que se han construido las operaciones básicas sobre la estructura, es más sencillo construir programas complejos.


CUADRO 1

sión correcta. El grafo correspondiente definía una expresión como una secuencia de al menos un término, si a éste le acompañaba otro, debería hacerlo sí, entre ambos, mediaba las operaciones suma o resta. En el cuadro 3 se puede observar cómo la traducción del grafo se corresponde con la descripción antes señalada. La definición de la variable de tipo conjunto *sum_res* tiene como objeto contener los diferentes símbolos terminales que contiene la definición. El procedimiento *salta_blancos* y el incremento de la variable *apunta* obligan a obtener el siguiente elemento a tratar. Eso solamente ocurrirá cuando se reconozca un símbolo terminal (como + y -). Puesto que para que exista un símbolo terminal es necesario que quede al menos un elemento de la cadena, también será necesario comprobar si la variable *apunta* no ha sobrepasado la longitud de la cadena.

Las ideas que se han desarrollado para la implementación del anterior procedimiento son también válidas para el procedimiento *termino*, de similares características.

La implementación del grafo correspondiente a factor (el procedimiento tiene igual nombre) representa una grafo del primer tipo mostrado en el cuadro 1. En él se combinan de forma mutuamente excluyente varios símbolos terminales y un símbolo no terminal, por lo que éste procedimiento posee sus correspondientes *if's* anidados, para determinar el tipo de símbolo que se va a tratar a continuación. La definición de un factor venía a decir que o bien era un número, o bien una expresión entre paréntesis, o bien un factor precedido de un signo.

Es importante darse cuenta que lo anterior es posible, debido a que se ha definido en el grafo el símbolo no terminal número como una secuencia de caracteres que debe empezar por un dígito. Si se hubiera incluido opcionalmente a dicha secuencia un signo inicial, no sería posible reconocer si a continuación va un factor (símbolo no terminal) o bien un número (también, símbolo no terminal), pues existiría la posibilidad de que ambos comiencen por un signo más o signo menos.

Cabe destacar que el procedimiento *factor* es recursivo, pues de haber un signo se procede a llamar, de nuevo, al mismo procedimiento. La parada de este procedimiento está asegurada, pues se avanza el puntero que indica el carácter que se está analizando actualmente, antes de proceder a la llamada, y una condición de parada de éste procedimiento es que dicho puntero sobrepase la longitud de la cadena.

Pero los procedimientos *factor* y *expresión_simple* también son mutuamente recursivos, y también lo son indirectamente con el procedimiento *termino*. De ahí la declaración previa utilizando la directiva *forward*. Ésta indica al procedimiento *factor* que para ejecutarse necesita de la existencia del procedimiento *expresión_simple*, que se encuentra descrito más adelante.

Una expresión será sintácticamente correcta cuando el puntero que indica el elemento de la expresión que se está analizando supere en uno la longitud de la cadena. Cuando se detecta un error mientras se procesa una expresión, se llama al procedimiento *error_anal_exp*. Este procedimiento acepta la posición donde se produjo el error y emite un mensaje apropiado dependiendo del número de error que se le pase como segundo parámetro, puesto que posteriormente no se podrá generar el árbol que represente a la expresión si se ha

Gráfo 1.-

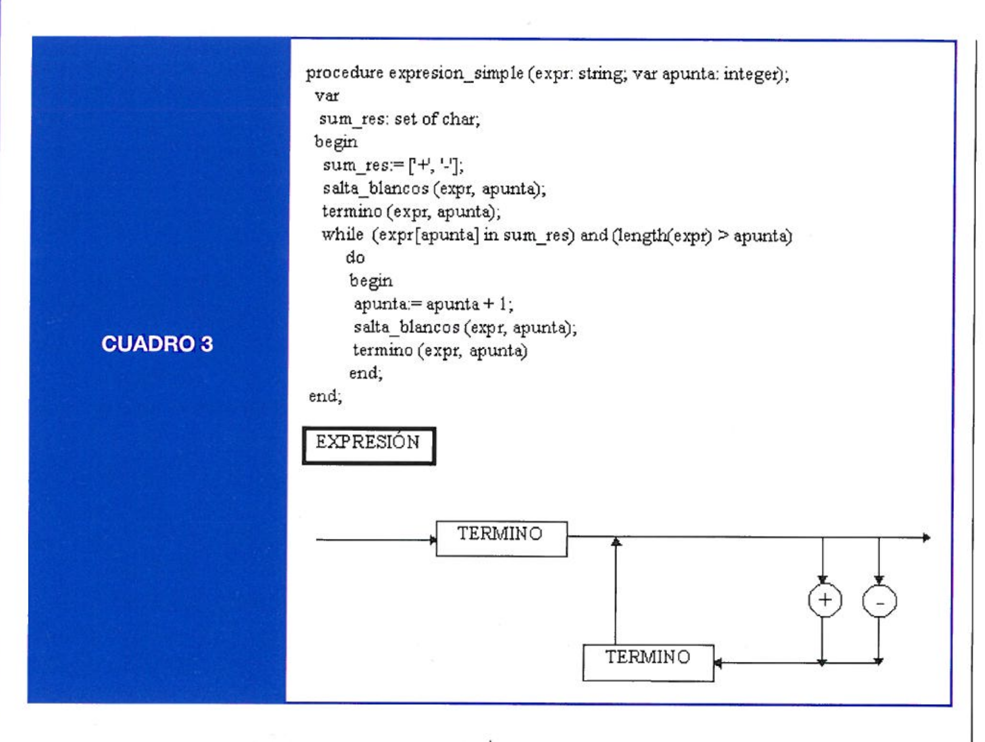
```
Si símbolo_terminal_1
  entonces
    siguiente_elemento_expresión
    reconoce_símbolo_no_terminal_1
  en otro caso
    Si símbolo_terminal_2
      entonces
        siguiente_elemento_expresión
        reconoce_símbolo_no_terminal_2
      en otro caso
        reconoce_símbolo_no_terminal_3
    fin-Si
fin-Si
```

Gráfo 2.-

```
reconoce_símbolo_no_terminal_1
mientras símbolo_actual = símbolos_terminales
   siguiente_elemento_expresión
   reconoce_símbolo_no_terminal_2
fin-mientras
```

CUADRO 2

CUADRO 3



```
procedure expresion_simple (expr: string; var apunta: integer);
 var
  sum_res: set of char;
 begin
  sum_res:= ['+', '-'];
  salta_blancos (expr, apunta);
  termino (expr, apunta);
  while (expr[apunta] in sum_res) and (length(expr) > apunta)
     do
      begin
       apunta:= apunta + 1;
       salta_blancos (expr, apunta);
       termino (expr, apunta)
      end;
end;
```


decidido interrumpir directamente el programa mediante la instrucción *halt*. Si se cree que ésta medida es demasiado extrema, se puede aumentar en uno el número de parámetros que acepta cada uno de los procedimientos que intervienen en el reconocimiento sintáctico. Este nuevo parámetro podría ser de tipo entero. Si tuviese valor 0, por ejemplo, representaría que el proceso de análisis es correcto hasta el momento. Por el contrario, si es distinto de cero se procede a parar el análisis y a visualizar el mensaje avisando del error encontrado en la expresión.

DE CADENA A LISTA

La transformación de una expresión que es sintácticamente correcta en una lista es un proceso que se podía haber realizado junto con el análisis sintáctico, pues se podrían haber aprovechado varios procedimientos. Pero por motivos de claridad se ha preferido que sea un proceso independiente. El procedimiento *cadena_a_lista* se encarga de esta tarea.

Si se analiza la sintaxis utilizada para definir el símbolo no terminal factor, se descubrirá que los símbolos más y menos representan un signo, no la operación suma o resta. La diferencia estriba en que estas dos últimas operaciones afectan a dos subexpresiones, mientras que el signo indica una operación que afecta a una única subexpresión. Si se pretende construir un árbol binario en el que cada nodo que contenga una operación afecte a dos subexpresiones, es necesario reinterpretar el significado de los operadores + o - cuando actúan como signo. La solución dada a esta situación consiste, en primer lugar, en detectar cuándo éstos símbolos actúan como signos y no como los operadores suma y resta. La cuestión es sencilla. Basta con entender los grafos: "si + o - van precedidos de los símbolos (, *, ó /, se trata de signos, en cualquier otro caso, se trata de los operadores suma o resta".

Una vez establecida la diferencia, se ha decidido simplificar una secuencia de operadores signo. Así, si en una secuencia hay un número impar de signos menos, combinada con cualquier otra secuencia de signos más, la secuencia de signos se simplificará a menos. En cualquier otro caso, la secuencia se simplificará a más.

Pero todavía falta por transformar un signo en una operación binaria que a efectos de evaluación siga proporcionando el mismo valor que la expresión original. Así, un signo seguido de la subexpresión a la que afecta se interpretará como la suma o la resta del número cero con dicha subexpresión, obteniéndose la operación binaria que se deseaba. Todo este proceso se realiza cada vez que en la cadena se detecta la presencia de un símbolo + o -, pero habrá que esperar a la generación del árbol para insertar el número 0, en este proceso solamente se procede a la simplificación de signos.

El proceso para generar la lista comienza con la eliminación de los caracteres blancos de que disponía la cadena de caracteres original. Esto permite agilizar el proceso de creación de la lista. No se había realizado con anterioridad debido a que expresiones como la siguiente (2 3*5), que es incorrecta, al eliminar los blancos se hubiera transformado en la siguiente cadena correcta (23*5), hecho que no se produce cuando una cadena es sintácticamente correcta. A continuación, mientras no se llegue al final de la cadena, se trata el carácter actual. Si éste es *, (,), /, se obtiene un nodo, se almacena como operación ,y se inserta al inicio de la lista. Si se trata de + o -, entonces se procede de la forma descrita en el párrafo anterior. Por último, si el carácter actual es un número, se procede a analizar los siguientes elementos de la cadena y se procede a almacenar todos los caracteres que pertenecen al número en una cadena auxiliar y a transformarlos en un literal numérico. Este proceso lo lleva a cabo el procedimiento *trans_cad_num*, utilizando para ello la función *val*, que dada una cadena de caracteres la transforma a formato numérico si es posible. En caso contrario, devolverá un código de error (se procede con este error de forma similar que en el resto de los errores). Una vez obtenido el literal numérico, se obtiene un nodo, se inserta como número y por último se inserta al inicio de la lista.

Todo este proceso, si ha sido correcto, producirá una lista que contiene todos los elementos de la expresión, pero dispuestos en orden inverso. Pero gracias a que la unidad *u_lista* proporcionaba un procedimiento (*invierte_lista*) para invertir una lista, se procede a invocarla. Es importante tener en cuenta que la utilización de las listas y los nodos se produce a través de los procedimientos y funciones que pertenecen a las unidades ya programadas, hecho que facilita el desarrollo del programa actual.

DE LISTA A ÁRBOL

Para realizar la evaluación de una expresión no es suficiente con tenerla almace-

nada en una lista, pues la evaluación no se produce de una forma lineal, debido a que las operaciones tienen distintas prioridades y la utilización de paréntesis modifica la prioridad de evaluación de las subexpresiones. Una forma de representar estas prioridades de forma implícita en una estructura de datos consiste en almacenar la expresión en un árbol, de tal forma que las operaciones que tengan mayor prioridad se encuentren situadas en los niveles más bajos del árbol. El procedimiento *lista_expr_arbol* realiza esta tarea.

El funcionamiento del anterior procedimiento es muy sencillo. Para cada nodo de la lista se procede a diferenciar el tratamiento entre un número o una subexpresión entre paréntesis y una operación o un signo. Si el nodo contiene un número, éste se almacena en un nodo auxiliar y se espera hasta que se conozca la operación que le seguirá. Si el nodo contiene una operación o un signo, se procederá a la inserción en el árbol según la prioridad que tenga con respecto a la raíz del árbol. En cada momento, salvo que el árbol esté vacío, hay dos puntos de inserción posibles:

- que el árbol actual sea el hijo izquierdo del nuevo nodo a insertar y el nodo que contiene el operando aún no insertado el hijo derecho del punto de inserción más profundo.

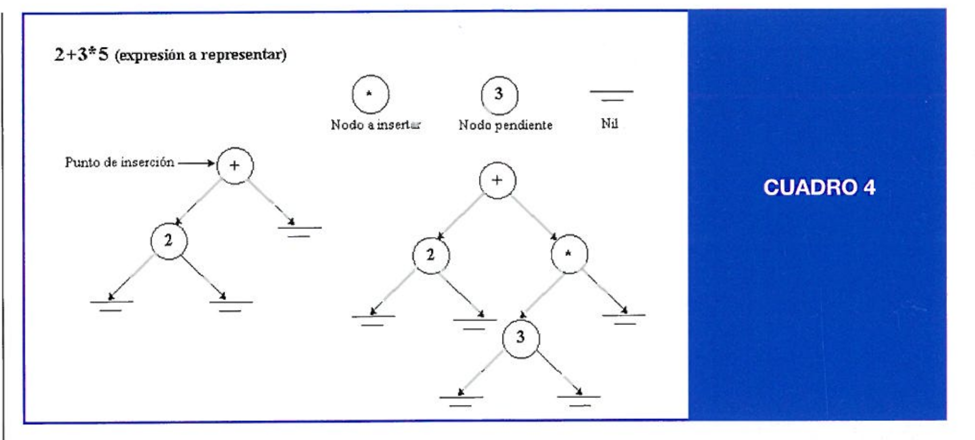
- que haya un nodo cuyo hijo derecho es nil, en cuyo caso el nuevo nodo sería el hijo derecho del anterior nodo y el hijo izquierdo del nodo a insertar sería el operando que estaba esperando.

El primer punto de inserción será seleccionado cuando la prioridad del nodo a insertar sea menor o igual que la prioridad del nodo raíz. En cualquier otro caso se seleccionará el segundo punto.

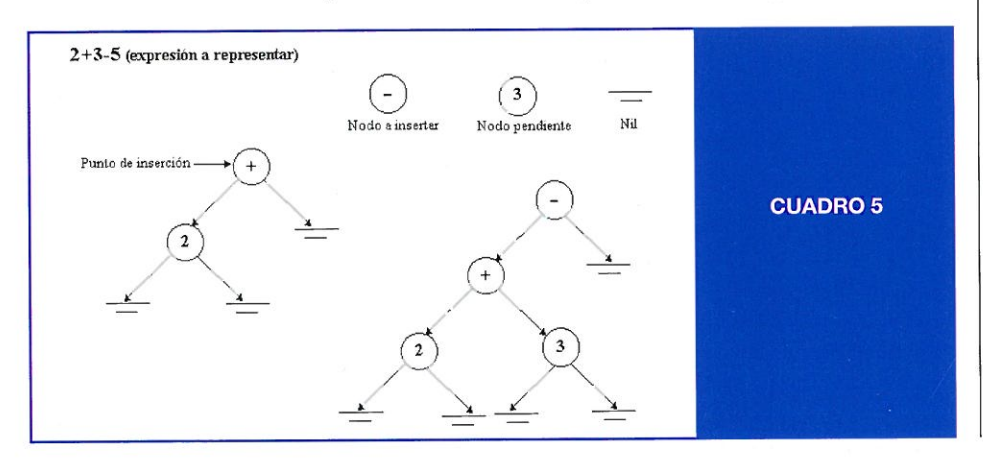
Para realizar el cálculo de prioridades se utiliza el procedimiento *prioridad*. Este, además de conocer la operación, nesita saber qué elemento es el predecesor del actual, para así poder diferenciar entre un signo y las operaciones suma y resta, pues ambos tienen diferentes prioridades. Pero además, en el caso de un signo, puesto que éste sólo afecta a la subexpresión que tiene a continuación,no dispone de un nodo pendiente para su inserción (se trata de una operación unaria). Por tanto, habrá que crear un nodo "ficticio" para transformar los signos en sumas y restas. Esto se logra creando un nodo que contenga el número 0 y que hará de primer operando, siendo la subexpresión que sigue al signo el segundo operando. Además, hay que asegurar que una vez insertado el signo éste conserve su prioridad. Esto se consigue almacenándola en una variable auxiliar.

En cuanto al tratamiento de las subexpresiones entre paréntesis, bastará con llamar al propio procedimiento para obtener el árbol correspondiente a la subexpresión encerrada entre paréntesis.

Los diferentes procesos de inserción se pueden observar en los cuadros 4 y 5.



CUADRO 4



CUADRO 5

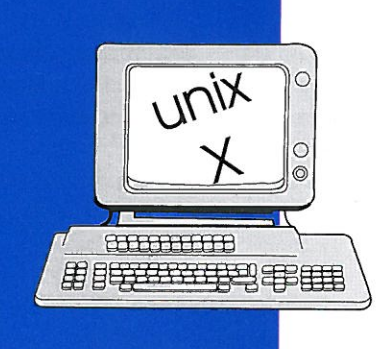
LA EVALUACIÓN

El proceso de evaluación lo realiza el procedimiento *evalua*. Este acepta como parámetros el árbol que contiene la expresión, un acumulador donde se almacenarán los resultados intermedios, una variable de tipo lógico que indicará si se ha producido un desbordamiento o algún error en las operaciones (división por cero) y, por último, otra variable de tipo árbol, que en caso de error mostrará en pantalla la subexpresión que provocó el error.

La forma de trabajar de éste procedimiento es muy sencilla. En primer lugar se comprueba si ha habido algún error en un resultado intermedio. Si es así, no hará nada. En caso contrario, se comprueba si la raíz del árbol contiene un número, en cuyo caso se devolverá dicho valor como evaluación del árbol. Si la raíz contiene un operador, entonces se llama recursivamente al subárbol izquierdo y al subárbol derecho y, tras comprobar que éstos han devuelto un valor correcto (a través de la variable booleana), se procede a realizar la operación indicada por la raíz del árbol sobre los valores devueltos por la llamada recursiva. Las operaciones se realizan a través de los procedimientos *suma*, *resta*, *producto* y *division*. Si tras realizarse la operación se produce un error, se proecede a asignar al parámetro *subexpr*, la subexpresión que ha provocado el error.

REDES TCP/IP: ARQUITECTURA Y PROTOCOLOS

Fernando J. Echevarrieta



Continuando con la explicación del funcionamiento interno de la red Internet, se presenta en este artículo la familia de protocolos asociados a la arquitectura TCP/IP, a la vez que se introducen algunos conceptos como el de datos fuera de banda, que se emplearán en los programas de próximas entregas.

La primera vez que uno oye hablar de Internet, casi siempre es en relación con el nombre de "TCP/IP". En muchos casos se habla de "TCP/IP" como "el protocolo que se usa en Internet" o "que hace funcionar Internet". Como explicación coloquial es válida, pero cualquier lector de esta publicación sabrá ya que IP y TCP son dos protocolos con funciones específicas y diferenciadas. Pero cuando se habla de TCP/IP no se está hablando únicamente de estos dos protocolos, sino de toda una familia, con fines muy diversos, que han constituido la arquitectura de la actual red Internet. Entre ellos, por supuesto, se encuentra el fundamental, el Internet Protocol o IP, encargado de generar la internet virtual [Echeva-1]. TCP, como se verá más adelante, proporciona el nivel de transporte más empleado, pero es posible encontrar otros protocolos de transporte en internet como UDP. Otros protocolos vienen a servir de apoyo a éstos, como ICMP. Y para realizar una comunicación, será necesario resolver muchos otros problemas como la correspondencia entre direcciones físicas y lógicas, para lo que se emplean protocolos como ARP, RARP y BOOTP; el mantenimiento de una información de rutas consistente, a través de protocolos de encaminamiento como GGP, EGP, OSPF o RIP; la gestión de red, para lo que se definen otros protocolos como SNMP; y la accesibilidad de una serie de servicios finales en forma de aplicación a través de otra serie de protocolos como TELNET, FTP, TFTP o SMTP. A todos ellos, y a otros muchos, se alude cuando se menciona "TCP/IP" de forma genérica, y el conocer de manera sucinta la función de algunos de ellos será el propósito del presente artículo.

NECESIDAD DE UNA ARQUITECTURA DE PROTOCOLOS

Desde el momento en que surge la necesidad de realizar una interconexión entre distintas redes, se fija como primer objetivo ocultar no sólo el hardware sino, también, el tipo de red sobre el que se sustenta la comunicación, para lo que se define una serie de servicios universales de comunicación.

Este objetivo de transparencia puede alcanzarse a través de los programas de aplicación que se encarguen de tratar en cada máquina con la red y el hardware específico con que ésta cuenta, proporcionando un sistema uniforme en la red. Sin embargo, este enfoque, si bien es más intuitivo y concreto y a veces se emplea en programas reducidos para resolver un problema bien definido, presenta graves problemas en cuanto el escenario aparece como un conjunto heterogéneo, más o menos numeroso y, sobre todo, cuando se persigue cierta generalidad. Así, si en determinado momento se desea incorporar una nueva funcionalidad, es necesario desarrollar código para cada diferente arquitectura. Por otra parte, el programador de aplicaciones debe enfrentarse a los problemas relacionados con la comunicación y no puede centrarse específicamente en su trabajo. Por ello, este enfoque no se suele emplear.

Otra de las posibilidades consiste en ceder esta labor a los sistemas operativos que faciliten una API de comunicaciones implementando interna-

mente una arquitectura de protocolos. Desde este punto de vista, se genera una red inter-red (internet, con minúscula) virtual interconectada a través de un protocolo encargado de generar esta imagen de red virtual, al que se denomina, por ello, protocolo de red [Echeva-1]. Esta arquitectura puede estar constituída por una serie de protocolos de propósito general, independientes de las aplicaciones, que se encarguen de transmitir fragmentos de información desde un origen a un destino sin importarles la naturaleza de la información que transportan. Por otra parte, al tratar los protocolos con fragmentos de información es posible realizar un tratamiento más eficiente. Con esta perspectiva, los programadores de aplicaciones emplean la API del sistema para realizar la comunicación necesaria sin necesidad de preocuparse por los mecanismos reales de esta comunicación, separando una problemática de la otra. Es este enfoque con el que se ha tratado a lo largo de los artículos en los que se ha realizado programación de comunicaciones, ya que se ha tratado directamente con llamadas al sistema operativo sin importar el tipo de red sobre el que se trabajaba (ni si ésta existía) y sin preocuparse por problemas de encaminamiento, congestión, pérdidas, duplicados, y otros aspectos de la comunicación. Es este planteamiento el que se emplea en todo sistema abierto y que se ha dado en llamar *internetworking*.

Cuando se habla de TCP/IP no se habla únicamente de dos protocolos, sino de una extensísima familia

ARQUITECTURA DE PROTOCOLOS TCP/IP

Como se ha expuesto en la introducción, la arquitectura de protocolos que constituye la red internet recibe el nombre de TCP/IP en honor a sus dos protocolos fundamentales: IP, el protocolo de red al que se aludía en el apartado anterior, y TCP, un protocolo de transporte fiable. Internet y la arquitectura TCP/IP se basan en un mecanismo de *internetworking* basado en cuatro premisas:

1. Los programas de aplicación no deben tener conocimiento del hardware para realizar una comunicación.
2. La interfaz de usuario debe ser independiente del sistema, lo que permite que los programas no necesiten conocer el tipo de red sobre el que se encuentran.
3. La internet no impone el uso de ninguna topología en especial [Recio-1].
4. El protocolo de red considera redes enteras como sistemas finales y permite transmitir información a través de redes intermedias que actúan como nodos intermedios en una red [Echeva-1].

El diseño es bastante consecuente. Así, del mismo modo que dos redes se encuentran físicamente conectadas si existe una máquina común que pase información de una red a otra, una red internet virtual conectará lógicamente redes a través de otras redes que, a cambio de pertenecer a la internet, deberán garantizar que colaborarán en la transmisión del tráfico de tránsito que necesite pasar a través de ellas, procedente de una red remota y se dirija a otra red remota. Este es el fundamento de algo que hoy en día ha llegado a ser tan natural como el hecho de que dos personas, sin conocimientos de comunicaciones, puedan jugar con un juego multiusuario a través de internet estableciendo una comunicación que podrá atravesar diferentes máquinas y redes con distintos sistemas operativos y que nunca llegarán a tener conocimiento del tipo de información que está pasando a través de ellos.

Este diseño, en el que se abstrae una red entera como un sistema final, unido al mecanismo de direccionamiento expuesto en números anteriores [Echeva-1][Echeva-2], en el que se presentaba las direcciones IP con una parte de red y otra de host, es lo que permite que el encaminamiento IP sea simple y eficiente y que los gateways o routers IP no deban ser especialmente potentes.

INTERNET E INTRANETS

A lo largo de los artículos, se ha venido escribiendo la palabra "internet" unas veces con mayúscula, otras con minúscula. Ésta, que parece una nimia cuestión, recientemente fue asunto de debate entre al coordinador técnico de la revista y el autor del artículo. En adelante, se pondrá especial cuidado en calificar como Internet, a la red mundial evolución de ARPANET o ARPA Internet, que con MILNET procede de la antigua DARPA Internet. Sin embargo, aún se continuará empleando el término internet, para referirse a toda red basada en *internetworking* mediante la arquitectura de protocolos TPC/IP. Y es que esta arquitectura ha demostrado ser tan eficiente que se está empleando en tantas redes aisladas de Internet como en la propia Internet. Es lo que se está poniendo de moda con el nombre

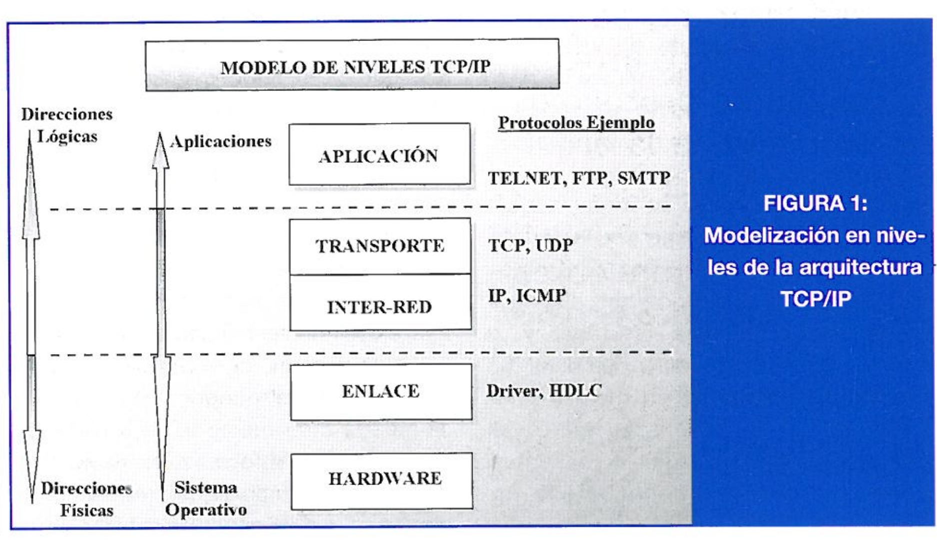


FIGURA 1: Modelización en niveles de la arquitectura TCP/IP

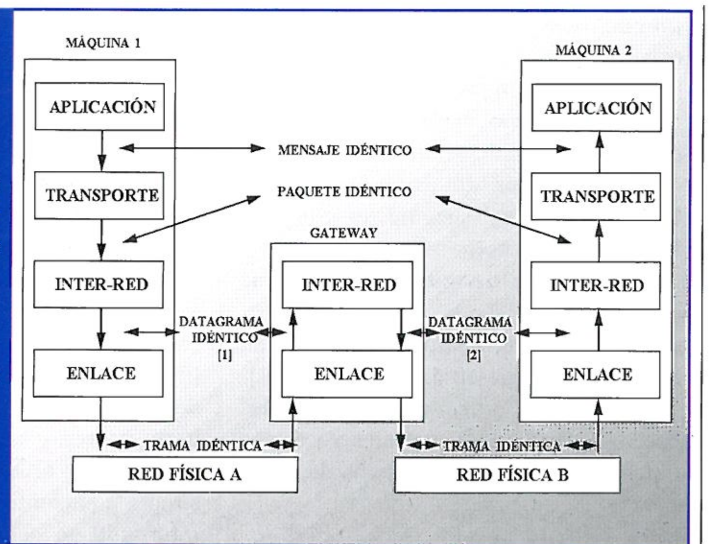


FIGURA 2: Esquema de una comunicación en internet a través de un nodo intermedio en la que se puede apreciar cómo únicamente a partir del nivel de transporte se realiza una conexión extremo a extremo

de intranets, redes corporativas, restringidas al ámbito de un grupo o empresa y aisladas en parte (a través de *firewalls,* por ejemplo) o totalmente de la Internet, pero cuyo funcionamiento es idéntico. El fenómeno ha crecido de tal forma que en este momento el número de servidores de Web en intranets es ya superior al que existe en Internet. Un ejemplo cercano de red "internet" es Infovía, una red basada en TCP/IP, por lo que el usuario puede acceder a ella del mismo modo que lo hace a Internet, sin necesidad de cambios en su sistema operativo y con el mismo software. Infovía, en cambio, no forma parte de Internet, aunque a través de ella se puede conectar con proveedores que servirán de *gateway* de salida a la Internet.

DESCOMPOSICIÓN EN NIVELES DE TCP/IP

Toda arquitectura de protocolos, al menos desde un punto de vista abstracto o de referencia, se suele descomponer en un modelo en capas o niveles en los que se sitúan diversos protocolos que especifican cómo transmitir la información entre máquinas. Esto obedece a una necesidad de atenuación de complejidad. El problema global se subdivide en distintas áreas con problemas aislados, y a cada una de ellas se asocia un nivel. El modelo de referencia en niveles más nombrado es el modelo de referencia OSI para la interconexión de sistemas abiertos, como ya se ha tratado en diversos artículos [Recio-2], propone una torre de 7 niveles. La arquitectura TCP/IP, que al contrario que OSI no proviene de ningún comité (como broma, en el mundillo se dice que un camello es un caballo elaborado por un comité), se puede considerar (a posteriori) dividida en cuatro niveles software y un quinto nivel que corresponde al hardware de comunicaciones. Si bien, como se irá mostrando, existe un cierto pararelismo en los niveles inferiores, se presentará a continuación la referencia de cuatro niveles TCP/IP y cuyo esquema se puede observar en la figura 1 (incluso, se ha "hecho encajar" algunos niveles OSI con la arquitectura TCP/IP, por lo que a lo largo de la serie se ha empleado y se empleará terminología OSI con fines didácticos). En cualquier caso, hay que dejar claro que esta clasificación es, más que nunca, un modelo, ya que si desde el punto de vista conceptual la descomposición en niveles aporta notables ventajas, introduce un considerable *overhead* a la hora de la implementación. Por ello, en el mejor de los casos se suelen romper los límites entre niveles mediante técnicas como la reserva de *buffers* en niveles superiores para acoger las cabeceras de niveles inferiores o el paso a niveles superiores de información como unidades máximas de transferencia con el objeto de optimizar la fragmentación.

IP es el protocolo responsable de la generación de una única red virtual

- Nivel de Aplicación

Constituye el nivel más alto de la torre TCP/IP. Al contrario que el nivel de aplicación OSI (verdadera pesadilla abstracta), se trata de un nivel simple en el que se encuentran las aplicaciones que acceden a servicios disponibles a través de internet invocadas por el usuario. Las aplicaciones pueden elegir el tipo de transporte que necesitan [Echeva-3], ya sea transmisión fiable de un flujo de bits o transmisión no fiable de mensajes delimitados, e interactúan con los protocolos de transporte únicamente para enviar y recibir datos. Es en este nivel en el que se han situado los artículos de programación de comunicaciones en los que se interactuaba con los protocolos de transporte a través de la interfaz *socket.*

- Nivel de Transporte

El nivel de transporte, análogo al nivel 4 OSI, proporciona una comunicación extremo a extremo entre programas de aplicación. En caso de emplear un transporte fiable, se gestionan retransmisiones y confirmaciones y se realiza control de flujo. El nivel de transporte divide los datos que le proporciona el nivel de aplicación en paquetes (terminología TCP/IP) que pasa al siguiente nivel junto con la dirección de destino. Como existe la posibilidad de atender a varias aplicaciones, añade información adicional a estos paquetes para identificar la aplicación remitente, la destinataria e información de *checksum.*

NOTA: El nivel de transporte intercambia mensajes delimitados o flujos de bits con su homólogo en la máquina remota (comunicación horizontal, entre entidades homólogas) y no paquetes, que es lo que pasa al nivel inferior (comunicación vertical, entre niveles en una misma máquina) [Recio-2].

- Nivel Inter-red (internet)

El nivel Inter-red es el que genera la red virtual internet. Para ello coloca la

información que le proporciona el nivel de transporte en datagramas IP (terminología TCP/IP), que añaden unas cabeceras con información necesaria para el nivel, y los pasa al nivel de Interfaz de red. Es en este nivel donde se emplean los algoritmos de encaminamiento. Por ejemplo, al recibir un datagrama del nivel inferior, decide en función de su dirección de destino si debe procesarlo y pasarlo al nivel superior o si por el contrario debe realizar un *forwarding*, es decir, si debe dejarlo "pasar a través" y retransmitirlo, colocándole una nueva dirección de destino y devolviéndolo al nivel inferior. También será este nivel el que procese los mensajes de error y control del protocolo ICMP, que circulan encapsulados en datagramas IP.

- Nivel de Enlace

Este nivel se limita a recibir del nivel superior los datagramas IP y transmitirlos por la red local, por lo que actúa como una interfaz de acceso a red. Puede ser tan simple como un driver de dispositivo o todo un subsistema con un protocolo de enlace propio como HDLC *(High level Data Link Control)*.

PROTOCOLOS DE RESOLUCIÓN DE DIRECCIONES

Como se ha venido destacando, con una interconexión de redes se genera una red virtual en la que las máquinas se identifican por una dirección de red lógica. Sin embargo, a la hora de transmitir información por un medio físico, se envía y recibe información procedente de una dirección física. Un diseño adecuado implica que la dirección lógica debe ser independiente de la física, pero como en última instancia la información se envía o proviene de los niveles inferiores que tratan con direcciones físicas, es necesario establecer un sistema que relacione uno y otro tipo de direcciones manteniendo su independencia. Así, en una internet es posible cambiar un ordenador de red, con lo que su dirección IP también cambiará, pero seguirá conservado la dirección hardware que se incluye "cableada" en su dispositivo físico de acceso a red. Del mismo modo, debe ser posible cambiar este dispositivo físico (por ejemplo, sustitución de una tarjeta ethernet) en un ordenador, con lo que su dirección hardware cambiará, sin necesidad de que cambie su dirección IP.

- ARP

Cuando una máquina necesita ponerse en contacto con otra, parte de su dirección IP conocida, pero será necesario conocer su dirección física para realizar el envío físicamente. Así pues, es necesario que el software de comunicaciones pueda realizar una "resolución" de direcciones lógicas a físicas y que ésta sea "dinámica" debido a que la relación entre direcciones lógicas y físicas puede ser cambiante. Para ello se ha definido el protocolo ARP *(Address Resolution Protocol)*. Cuando una máquina desea saber la direción física correspondiente a una dirección IP, envía por broadcast una petición ARP. El protocolo establece que una máquina debe contestar a una petición ARP sólo si ésta lleva su dirección IP. Por lo tanto, solo contestará aquella máquina que corresponde a la dirección IP buscada con un mensaje que transportará la dirección física. El software de comunicaciones llevará una caché con las últimas relaciones dirección IP / dirección física empleadas. De este modo, la siguiente vez que se necesite contactar con la misma máquina no será necesario preguntar su dirección física, con lo que el envío de mensajes ARP queda reducido a la mínima expresión.

Una intranet no es otra cosa que una red privada con la arquitectura TCP/IP

- RARP

En ciertas ocasiones, el problema surge a la inversa: una máquina debe conocer su propia dirección IP, y para ello la única información de la que dispone es la correspondiente a su dirección física. Este era un caso común en las estaciones de trabajo sin disco, que arrancaban a través de red. Hoy en día, en que los discos son suficientemente baratos, sigue surgiendo esta necesidad por motivos, por ejemplo, de seguridad o de mantenimiento de una red en que todas las configuraciones son iguales y sus características diferenciales, como la dirección IP, se almacenan centralizadas en un servidor. También es el caso de las máquinas que cada vez que arrancan disponen de una dirección IP diferente, como es el caso de cualquier PC que accede a través de modem a un proveedor de internet del que obtiene, en cada conexión, una dirección IP diferente de entre las que en ese momento están sin utilizar. Esto permite que el proveedor disponga de muchas menos direcciones de las que luego utilizará.

Sea como fuere, se dispone de una máquina que no conoce su dirección IP, por lo que no se puede comunicar a través de la internet, pero sí puede hacerlo a través de su red local mediante su nivel de enlace. Así, basta con que envíe por broadcast en su red física una petición RARP *(Reverse Address Resolution Protocol)* que contendrá su dirección física para que un servidor que esté escuchando en la misma red y disponga de una tabla de correspondencias (en UNIX /etc/ethers), le pueda enviar una respuesta que contenga la dirección IP buscada.

- BOOTP

El protocolo RARP resuelve el problema de la resolución inversa de direcciones. Sin embargo, no es todo lo eficiente que podría ser, ya que en el mensaje de vuelta se podría incluir más información que la simple dirección IP del solicitante. Así pues, existe otro protocolo, BOOTP *(BOOTstrap Protocol)*, que además de la dirección IP del solicitante, facilita la de un *gateway* en su red, la de un servidor, y puede proporcionar otra información adicional. Todo ello faci-

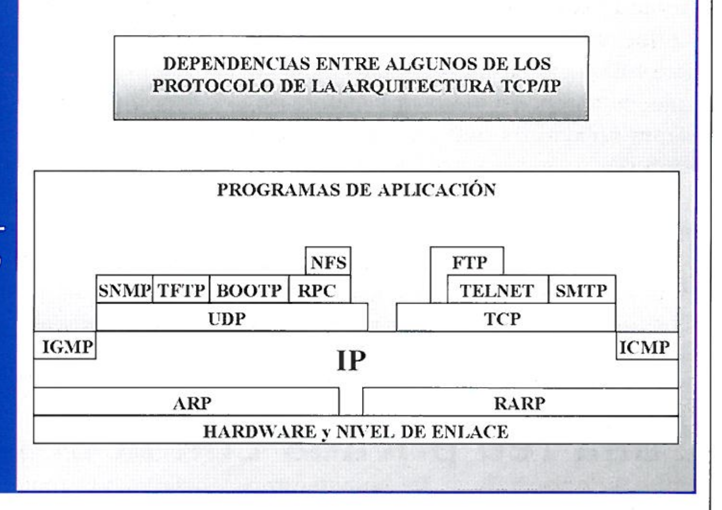


FIGURA 2: Esquema de depencias entre los protocolos tratados a lo largo del artículo.

lita enormemente la movilidad y el mantenimento de las máquinas. El protocolo BOOTP, paradójicamente, se sustenta sobre UDP, encapsulado en IP antes de conocer la dirección IP origen. Esto se debe a que, como se mencionaba en [Echeva-1], es posible emplear la dirección IP de broadcast local 255.255.255.255 antes de que IP conozca la dirección IP origen.

NIVEL INTER-RED: IP

La red virtual internet se genera en el nivel inter-red (internet). Este nivel facilita la interconexión entre redes, pero ésta interconexión no es directa entre extremos, labor del nivel de transporte, sino salto a salto. Es decir, es en este nivel donde se realiza la labor de encaminamiento. Esta labor consiste en decidir en función de la dirección IP destino de la información recibida, si ésta debe ser procesada localmente o debe ser enviada a otro nodo y cuál debe ser éste para que finalmente llegue a su destino.

- IP

La estrella del nivel inter-red, o nivel de red en terminología OSI, es el protocolo IP. IP es un protocolo de datagramas, es decir, no orientado a conexión, con mensajes con un tamaño máximo. Cada datagrama IP se gestiona de forma independiente, por lo que dos datagramas que formen parte de la misma comunicación pueden llegar a su destino por rutas distintas, lo que puede provocar que lleguen en distinto orden del que salieron, que se pierda alguno o, incluso, que lleguen duplicados. El protocolo IP es no fiable, es decir, no se encarga de corregir estos problemas ni tampoco de informar de ellos o detectarlos.

Los datagramas IP contienen una cabecera y un cuerpo de datos. Además de los datos que provienen del nivel superior, incorporan información de fragmentación, precedencia, *checksums*, direcciones IP origen y destino. Para ello emplea campos de longitud fija y un campo de opciones de longitud variable.

Dado que IP es "la sangre de Internet", se estudiará en profundidad en un próximo artículo.

- ICMP

Para el correcto funcionamiento de una internet es necesario disponer de un mecanismo de comunicación de información de control o de errores, de la misma forma que en un organismo vivo existe una transmisión de impulsos nerviosos que pueden dar lugar a ciertos actos reflejos. Para ello, se emplea el protocolo ICMP *(Internet Control Message Protocol)* que suelen emplear los *gateways* y los hosts para notificar a los remitentes de datagramas condiciones especiales de la red, como la existencia de una congestión; o de error, como la detección de bucles, la inalcanzabilidad de un host y la petición de cambios de ruta. También se emplea este protocolo para realizar peticiones de *echo* que notifiquen la alcanzabilidad de una máquina, lo que se conoce habitualmente como protocolo de *ping*, que no es tal protocolo sino una interfaz de ICMP.

Los mensajes de ICMP viajan encapsulados en el cuerpo de datos de los datagramas IP.

- IGMP

Otro protocolo íntimamente ligado a IP es el IGMP *(Internet Group Management Protocol)*. En concreto, se emplea en aquellas máquinas y *gateways* que emplean IP multicast. El IP *multicast* es una variante de IP que permite el empleo de datagramas con múltiples destinatarios, para lo que se emplean direcciones de clase D. Las máquinas que participan en una comunicación *multicast* se agrupan en conjuntos dinámicos y emplean el protocolo IGMP para notificar entradas o salidas de estos grupos a los *gateways*, así como otra información de control de manera muy parecida a como lo hace ICMP. Los mensajes IGMP, al igual que los ICMP viajan encapsulados en datagramas IP.

NIVEL DE TRANSPORTE

El nivel de transporte se encarga de proporcionar una comunicación extremo a extremo. A partir de este nivel, la información que se pasa desde un nivel superior al inferior en la máquina que envía será idéntica a la que en su homóloga remota, que hace de receptora, se pasará del nivel inferior al superior. Como se puede observar en la figura 2, esto no ocurre con el nivel de red, ya que los datagramas que pasan del nivel de inter-red al de enlace en la máquina 1 son iguales a los que recibe el nivel inter-red del *gateway*, y análogamente son iguales los enviados por éste y los recibidos por la máquina 2. Sin embargo, los dos pares son diferentes entre sí ya que, como mínimo, varían en sus direcciones IP origen y destino. Así pues, el datagrama generado en la máquina 1 y el recibido en la 2 son diferentes. Sin embargo, la información correspondiente al nivel de transporte es igual en ambas máquinas.

En el nivel de transporte, se emplean dos protocolos: UDP y TCP.

- UDP

UDP proporciona un nivel de transporte no fiable de datagramas, es



decir, apenas aporta semántica a IP salvo por el hecho de facilitar la comunicación extremo a extremo, lo que realiza a través de la idea de puerto que ya se ha empleado en los artículos dedicados a programación basada en *sockets.*

- TCP

TCP es la otra "estrella de la función", ya que proporciona un transporte fiable de flujos de bits entre aplicaciones. TCP es realmente un protocolo de propósito general, que puede funcionar sobre una red sin necesidad de IP. Está pensado para el envío de grandes cantidades de información liberando al programador de aplicaciones de la difícil gestión de fiabilidad (retransmisiones, pérdidas, duplicados, orden) que gestiona el propio protocolo. Esta gestión, tarea compleja, implica un coste en eficiencia, por lo que se suele emplear UDP cuando es importante la velocidad y no preocupa la fiabilidad con lo que, practicamente en ningún caso, la gestiona una aplicación. TCP es un protocolo orientado a conexión, por lo que establece circuitos virtuales. Consigue una gran eficiencia mediante la concatenación y/o partición de los datos que recibe del nivel superior decidiendo el momento más adecuado para transmitirlos, aunque también permite la posibilidad de realizar un push de los datos para enviarlos en un momento determinado. Establece conexiones *full-duplex* (bidireccionales simultáneamente) que hace ver a las aplicaciones como dos flujos independientes de comunicación con sentidos opuestos, aunque en cualquier momento se puede cerrar uno de los dos sentidos y convertirla en *half-duplex.* Otra de las posibilidades que ofrece es la de transmisión de datos fuera de banda, es decir, datos que "se saltan la cola" para llegar a destino antes que otros que salieron anteriormente. De esta forma se transmiten órdenes como la de interrupción de un proceso al pulsar CTRL+C en una conexión de terminal y se hace posible la interrupción de procesos que aún no han leído datos que tienen a la espera. Dada la importancia de este tipo de datos, se estudiará su manejo desde la interfaz *socket* en un próximo artículo.

Es necesaria la existencia de protocolos de resolución de direcciones para asociar direcciones físicas y lógicas

NIVEL DE APLICACIÓN

Entre los servicios que caracterizan a la Internet como tal, se encuentran muchos sustentados por protocolos que se encuentran en el nivel de aplicación. Por ejemplo, dos de los estándares más empleados son TELNET, que proporciona un servicio de conexión remota con posibilidad de envío de comandos, y FTP *(File Transfer Protocol),* que proporciona un servicio extendido de transferencia de ficheros con posibilidades añadidas para el manejo de directorios. Pero FTP no es el único protocolo dedicado a transferencia de ficheros. En ocasiones anteriores ya se mencionó el NFS *(Network File System)* de SUN que se sustenta sobre RPC *(Remote Procedure Call),* definido como plataforma independiente que también puede ser empleado por otros programas de aplicación (como se hizo en su momento). Y para ocasiones en que es suficiente contar con un servicio de transferencia de ficheros sin necesidad de ninguna extensión, se emplea el TFTP *(Trivial File Transfer Protocol),* protocolo que se sustenta sobre UDP y reduce al mínimo la complejidad y el *overhead* asociados a FTP, lo que lo hacen ideal para su inclusión en EEPROMS de arranque en dispositivos de red junto con BOOTP.

Otra facilidad sin la que no se concibe la internet es el correo electrónico, sustentado sobre el protocolo SMTP *(Simple Mail Transfer Protocol),* que define un formato de mensajes de protocolo basados en un número de operación y un mensaje en texto ASCII legible por los humanos. De esta forma, conociendo el protocolo, es posible conectarse mediante TELNET al puerto adecuado y "hablar" SMTP interactivamente por teclado.

BIBLIOGRAFÍA Y REFERENCIAS

- Comer, Douglas E. "Internerworking with TCP/IP. Volume I; Principles, Protocols, and Architecture"
- Tanembaum, Andrew S. "Redes de Ordenadores"

Las referencias corresponden a artículos publicados en Sólo Programadores por María Jesús Recio:
[Recio-1] "Redes Locales: Introducción y conceptos básicos" núm. 21
[Recio-2] osi, núm. 22
y por el autor de este artículo:
[Echeva-1] "TCP/IP:", núm 23
[Echeva-2] "Bases de datos de red", N 22
[Echeva-3] "Arquitecturas Cliente-Servidor: La interfaz socket (I)" núm 17

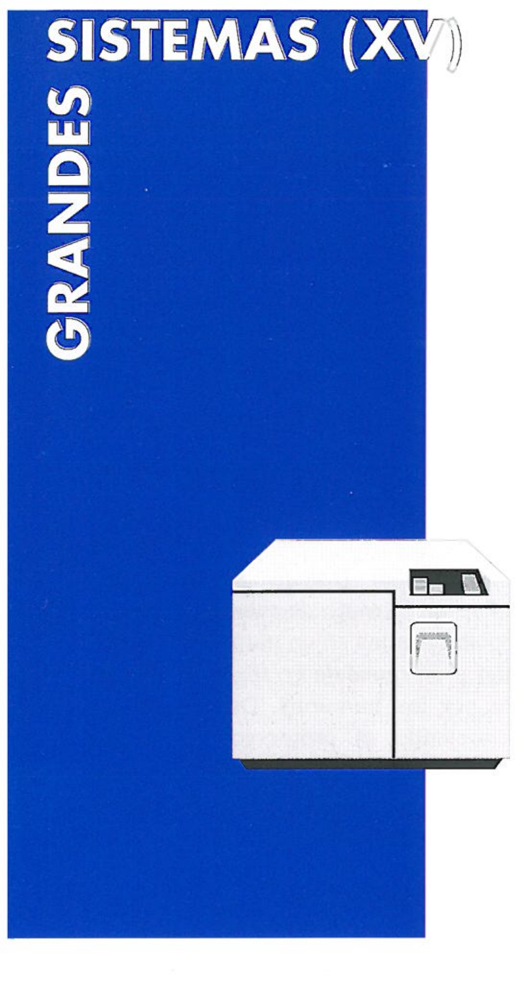
CONCLUSIONES

Como podrá apreciar el lector, el término "TCP/IP" comprende una familia de protocolos mucho más amplia de lo que pudiera parecer, y aún se han reservado para otras entregas los protocolos de encaminamiento y gestión de red que se mencionaban en la introducción. Pero para aquellos lectores a los que les parezca que el número de protocolos es grande tras la lectura de este artículo, se pueden mencionar otros a los que se ha aludido en anteriores números o que aún ni se han citado, como DOMAIN, ECHO, NTP, NETBIOS, DISCARD, CHARGEN, QUOTE, USERS, DAYTIME, TIME, NICNAME, PPP, POP3, NNTP, HOSTNAME, SFTP, FINGER, BGP, AUTH y otros tantos que han ido quedando obsoletos.

CONTACTAR CON EL AUTOR

Para cualquier duda, comentario, sugerencia o crítica, se anima al lector a que se ponga en contacto con el autor a través del correo del lector o, preferiblemente mediante correo electrónico a través de:

E-mail Internet: echeva@dit.upm.es
E-mail CompuServe: 100646,2456
WWW: http://highland.dit.upm.es:8000



OBJETOS NATURAL (III)

José María Peco

El pasado mes, una vez descritos de forma general los distintos tipos de objetos que componen este producto, el autor propuso un esquema mediante el cual llevaba a la práctica el principio básico de que todo programa debe tener una única entrada y una única salida, ya que considera al programa como un elemento transformador de información.

Los puntos del esquema ya tratados fueron:

- Definición de datos.
- Recepción de parámetros iniciales.
- Preparación inicial de datos.
- Presentación de un mapa con los datos iniciales que sirva de interface para el tratamiento de la información.
- Validación de datos.

Los siguientes puntos completan la exposición del esquema referenciado.

TRATAMIENTO DE LOS DATOS

Una vez validados los datos, se debe proceder a elaborar definitivamente la información. Este punto es muy variado y, dependiendo de que se trate de programas de consulta o de actualización, podrá contener procesos más o menos complejos.

Tanto en este paso del esquema como en el de las validaciones previas, se puede invocar a otros procesos, bien para validar, bien para obtener datos elementales en base a parámetros muy específicos. Así, por ejemplo, para validar el código de una divisa se invocaría a un subprograma, al cual se le pasarían como parámetros el código de divisa y un continente para que devuelva si es válido o no.

La figura 1 muestra los objetos referenciados en el fuente de una aplicación operativa mediante la utilidad LISPGM7P que acompaña al artículo.

La diferencia entre utilizar objetos tipo subprograma o subrutina estriba básicamente en dos hechos:

1.- Una subrutina comparte área global con el programa llamante, y además debe correr en la misma máquina en la que se esta ejecutando el programa llamante.

2.- Un subprograma puede correrse en una máquina distinta a la del programa llamante, y como consecuencia no comparte datos globales. En el caso de que se ejecuten ambos objetos en la misma máquina y el área global que utilizaran fuera la misma, sería considerada como distinta, siendo inicializada en el subprograma cada vez que fuera invocado este.

Así, en el ejemplo anterior, si el código de divisa dependiera del código de banco, y éste se encontrara cargado en el área global del programa principal, podría utilizarse una subrutina externa a fin de que ese dato global lo recogiera del área. Pero también podría usarse un subprograma, solo que en este caso los parámetros a pasar serían código de banco y código de divisa, así como un continente para que devolviera el resultado de la validación.

Por último, cabe reseñar que también se puede invocar un objeto tipo programa para hacer una validación o elaborar un dato elemental. Esta llamada se hace utilizando la sentencia *FETCH RETURN* . Esta opción se diferencia de las anteriores en:

- Los datos no los recoge el "programa llamado" en un área de datos *parameter*, tal y como se podía reali-

Este mes se cierra el tema de los objetos natural, mediante el cual su autor ha descrito las características de cada uno y propone, de forma muy personal, el uso que debe hacerse de los mismos, basándose en su experiencia y en sus conocimientos.

zar en los otros dos tipos, sino que se recogen vía *input* a través del *stack* como se comentó al hablar de las áreas de datos.

- Se comparte área de datos global, siendo esta otra forma de pasar datos al "programa llamado".
- La devolución de datos al programa llamante se debe realizar vía datos globales o vía *Stack*.

Como norma general se debe perseguir que los módulos sean lo más pequeños posible, ya que esto facilita su mantenimiento. A la hora de elegir el tipo de objeto a utilizar se deben tener en cuenta las características de cada tipo, y en caso de ambigüedad, elegir, de acuerdo con el cuadro de la figura 2, aquel que represente menor coste de llamada.

PROCESOS QUE NECESITAN PANTALLA DE CONFIRMACIÓN

Hay instalaciones en las que se requiere que antes de grabar datos se presente una pantalla con los datos elaborados y protegidos para solicitar al usuario que confirme la operación que se desea realizar.

Este requerimiento sigue ajustándose perfectamente al esquema dado, solo que en este caso, se debe:

- Hacer uso de una variable de control para proteger los campos.

Para poder hacer uso de la variable de control, esta debe estar definida tanto en el módulo llamante como en el mapa. Esta variable, si se desea que afecte a todo el mapa, se definirá en el mapa *set*, o mapa del sistema que sirve de interface para especificar los atributos de un mapa de usuario (se accede a él pulsando PF2 desde el editor de mapas), y deberá especificarse el nombre de la variable en el campo *Control var* (ver figura 3). Además, los campos de entrada y los campos de entrada/salida (o modificables) deberán tener definido el atributo "Y" (dinámico) en el campo de atributos AD (figura 4).

En el caso de que se definan variables de control a nivel de campo elemental, estas se definirán en el campo CV (figura 4). Este es el caso de la variable de control *C-COPYCODE(*)* asociada al campo *M-COPYCODE(*)* de la figura 4. Esta variable sirve para desproteger el campo *M-COPYCODE*, pero sólo cuando el campo *T-COPYCODE* tenga un valor. De esta forma se impide el elegir una acción (listar, imprimir, etc) cuando no se tiene un nombre sobre el que ejecutar la acción.

Figura 1

```
UTILIDAD JMPDES   <<<<<<<< Utilidad para desarrollo >>>>>>   13/05/96 18:13:30
                  Libreria: UTILIDAD Objeto: LISPGM1P
-----------------------------------------------------------------------------
 Programas invocados    Mapas    Subrutinas Subprogram  Areas D.    Copycode
---------------------- ---------- ---------- ---------- ---------- ---------- -
  MENU                 LISPGM1M                                    LISPGMER
  LISPGM4P
  LISPGM5P
  LISPGM6P
  LISPGM7P
  LISPGM8P

-----------------------------------------------------------------------------
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      AYUDA       SALIR       P:LP  M:LMP V:LVP                <<--- --->>
```

- Hacer la llamada al mapa dentro de un bucle, por ejemplo tipo *REPEAT*.

  En este caso, el esquema del programa sería el de la figura 5.
- Poner los puntos de validación de datos y elaboración de datos en sendas subrutinas internas, de modo que sólo se ejecuten cuando corresponda.

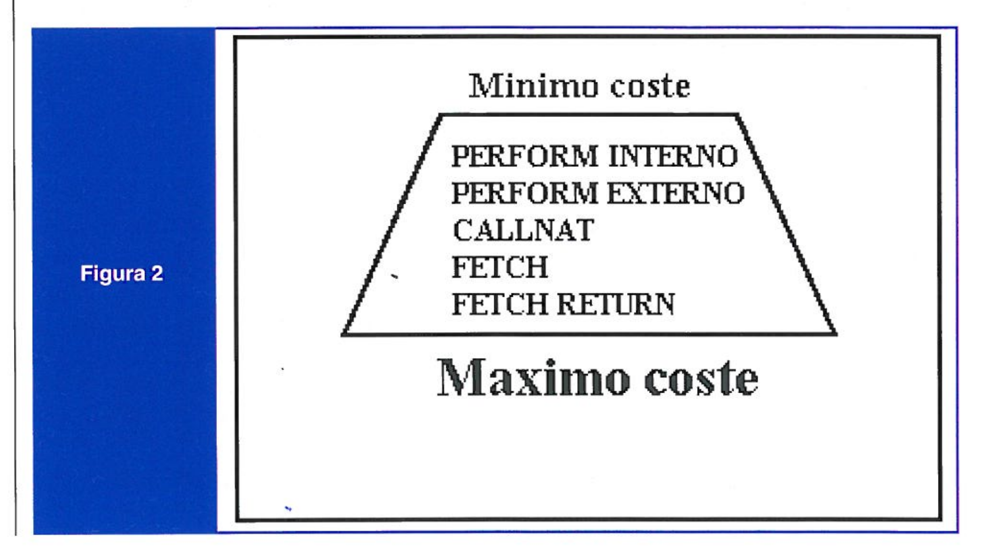
  Así, de acuerdo con el esquema anterior, la primera vez que se presenta el mapa no se puede pulsar PF10, pues no está activada, por lo que las únicas Pfs que se pueden pulsar son : PF3 para terminar el proceso y *ENTER* para validar.

  En el caso de que no se supere alguna validación, se regresaría directamente al mapa mediante *REINPUT*, por lo que solo se llegará a activar PF10 cuando se superen todas las validaciones. En este caso, además, se asigna el valor de protección a la variable de control, por lo que cuando vuelva a aparecer el mapa saldrán todos los campos protegidos, y en caso de que se pulse PF10 ya no es necesario repetir las validaciones, sino que sólo hay que realizar el tratamiento propio de la transacción.
- Hacer uso de la sentencias *END TRANSACTION* para pasar a definitivas las modificaciones realizadas en la base de datos, o de *BACKOUT TRANSACTION* para deshacerlas.

LOS OBJETOS NATURAL Y LA POO

Por último, el autor no quisiera terminar el artículo sin relacionar los objetos NATURAL con la Programación Orientada a Objetos tan en boga hoy en día. Por tanto, este apartado figura únicamente para establecer diferencias y evitar confusiones.

Así, desde el punto de vista del autor, en lugar de OBJETOS NATURAL, este artículo debería haberse titulado "TIPOS DE MÓDULOS EN NATURAL", con lo que se evitaría relacionar los objetos vistos con los objetos de la POO. Pero ha preferido usar la terminología del entorno para evitar confusio-



nes y aclarar ahora que en el entorno NATURAL se entiende por objeto el código ejecutable que resulta de compilar un fuente, mientras que en cualquier entorno de sistemas se entiende por módulo un "trozo" de codificación en lenguaje máquina, pero no totalmente resuelto, y por tanto no directamente ejecutable. Por decirlo de otra forma, un módulo siempre forma parte de algo mayor, pero en sí no es algo completo.

Por otra parte, si bien es verdad que una cosa es la modularidad y otra la programación orientada a objetos, también es verdad que entre ambas sólo hay un paso. Y NATURAL tiene dado un paso de gigante para llegar a la POO, pues realmente es un programa principal que ejecuta comandos del sistema y comandos creados por el usuario, con lo que en cualquier caso ya se ha interpuesto la capa necesaria para la POO entre el usuario y los datos. Ahora solo habría que hacer que los datos fueran sensibles a cualquier evento externo para que se ejecutara cualquier módulo de los comentados, pero esto, como el lector comprenderá, se sale de los objetivos de este tema.

DIAGRAMA DE FLUJO

La regla "Un programa, un mapa" permite discriminar los objetos de tipo programa, ya que estos serán los únicos que incluirán llamadas a mapas, aunque no todos los mapas tienen que incluir llamadas a mapas (Ejemplo: programas Batch).

Para transacciones On-line, al seguir esta regla, se pueden usar como documentación los esquemas modulares o diagramas de flujo con la simbología expresada en la figura 6.

Hay que tener en cuenta que si bien en cada librería se encuentran todos los módulos de una aplicación (gestión de almacén, por ejemplo) ésta se compone de muchas transacciones, entendiendo por tal un conjunto de objetos que permiten realizar una acción muy concreta dentro de una aplicación. Así, dentro de la aplicación de Almacén, una transacción seria Recepción de productos, la cual a su vez puede tener distintas opciones.

Otra consideración a tener en cuenta se centra en la nomenclatura utilizada

```
09:31:00                 Define Map Settings for MAP                       96-05-14

 Delimiters              Format                          Context
 -----------------       ---------------------------     ---------------------------
 Cls Att CD  Del         Page Size ...... 23             Device Check ....  ________
  T   D      BLANK       Line Size ...... 100            WRITE Statement    _
  T   I      ?           Column Shift ... 0 (0/1)        INPUT Statement    X
  A   D      _           Layout ......... ________       Help  'LISPGMOH'__________
  A   I      )            dynamic ....... N (Y/N)         as field default N (Y/N)
  A   N      ^           Zero Print ..... N (Y/N)
  M   D      &           Case Default ... UC (UC/LC)
  M   I      :           Manual Skip .... N (Y/N)        Automatic Rule Rank  1
  O   D      +           Decimal Char ... ,              Profile Name .... SYSPROF
  O   I      (           Standard Keys .. Y (Y/N)
                         Justification .. L (L/R)        Filler Characters
                         Print Mode ..... __             ------------------------
                                                         Optional, Partial .... _
                         Control Var .... ________       Required, Partial .... _
                                                         Optional, Complete ... _
                                                         Required, Complete ... _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit                                                 Let
```

Figura 3

por casi todas las instalaciones, mediante la cual se asigna el nombre de la transacción a los 4 o 5 caracteres iniciales del nombre de los objetos y reserva, de los 3 o 4 caracteres restantes, una posición fija para el carácter asociado al tipo de objeto, de acuerdo con la tabla de la figura 1 del primer artículo de este tema.

Pues bien, en el rectángulo superior, o raíz del árbol jerárquico que representa, figurará el nombre de la transacción. Y como generalmente cada transacción cuenta con un menú inicial, de la raíz reseñada saldrá una línea con las distintas opciones de dicho menú encerradas en un círculo.

A la derecha de estas opciones, otras líneas conducen a los rectángulos que representan los programas de cada una de ellas. Dentro de este rectángulo se describirá en 2 o 3 palabras la función del módulo (Alta Pedido), y en los distintos ángulos los sufijos que identifican los distintos módulos que acompañan a ese programa, tal y como muestra la figura.

De cada uno de estos módulos de nivel inferior también pueden salir nuevas opciones, por lo que, si esto fuera así, éstas se reflejarían en el correspondiente diagrama.

Cuando la bifurcación proviniera de una PF, ésta se representaría como una opción de menú, con la salvedad de ir encerrada dentro de un triángulo el número de la PF que conduce a otro módulo.

Los comentarios a la figura 6 son los siguientes:

(Opción): representa las distintas opciones en el caso de que la función presente un menú o lista seleccionable con distintas opciones.

PPP- Es el nombre del programa que se ejecuta. Puede ser Programa, Subprograma o Subrutina, pero siempre es el nombre del módulo principal de la función.

MMM- Es el nombre del mapa que es llamado en ese programa. Solo se puede asociar un mapa a una función.

NNN- Es el nombre de los subprogramas invocados o que se ejecutan en otra máquina.

CCC- Es el nombre de los COPYCODES que se encuentran incluidos en el fuente del módulo principal de la función. En el caso de que hubiera varios, se especificarán uno debajo de otro.

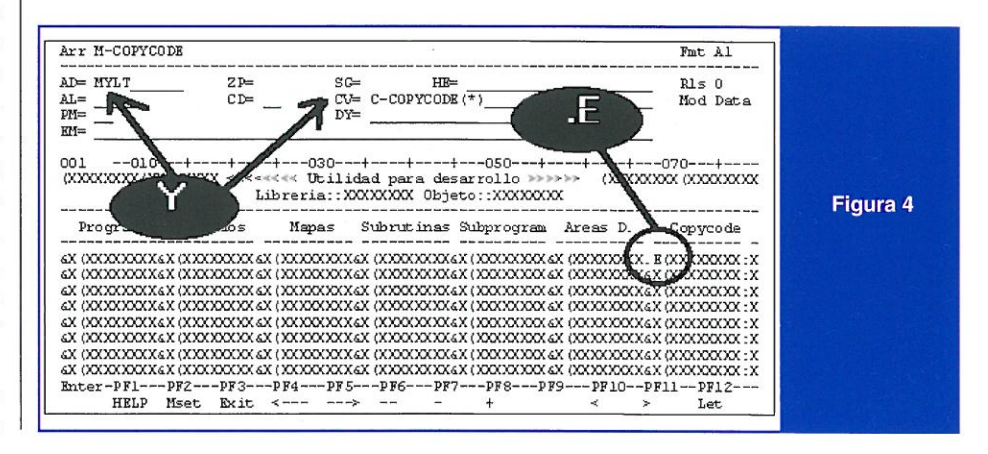
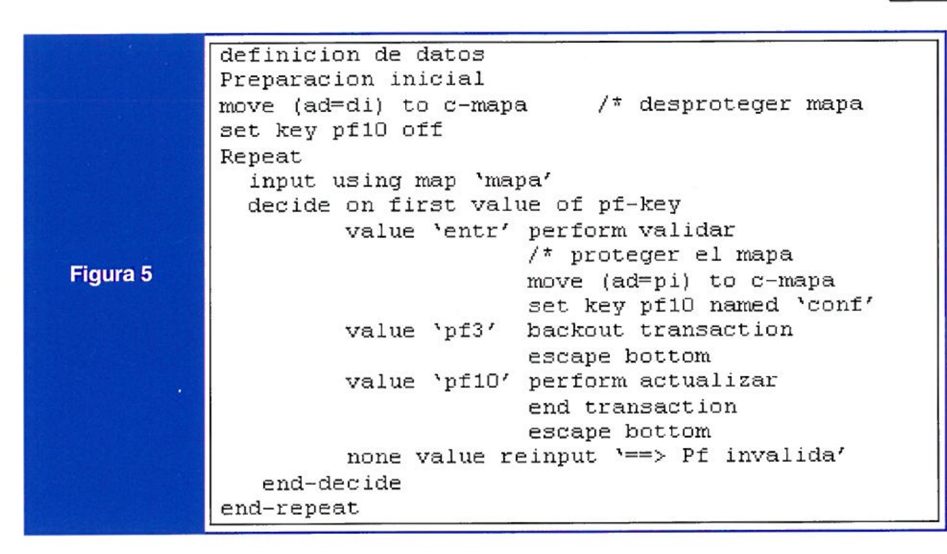


Figura 4



```
definicion de datos
Preparacion inicial
move (ad=di) to c-mapa      /* desproteger mapa
set key pf10 off
Repeat
   input using map 'mapa'
   decide on first value of pf-key
          value 'entr' perform validar
                       /* proteger el mapa
                       move (ad=pi) to c-mapa
                       set key pf10 named 'conf'
          value 'pf3'  backout transaction
                       escape bottom
          value 'pf10' perform actualizar
                       end transaction
                       escape bottom
          none value reinput '==> Pf invalida'
   end-decide
end-repeat
```

Figura 5

BUFFERS DE NATURAL

Como anexo al artículo, y con objeto de ampliar conceptos, se añade este apartado con la descripción de los distintos *buffers* usados por NATURAL.

A la hora de diseñar una aplicación es importante tener en cuenta los *buffers* que se definen en NATURAL por cada sesión, ya que existe una serie de áreas especiales o *threads* que son utilizadas por NATURAL de forma diferente, según se encuentre en tiempo de edición, compilación y ejecución.

- *USIZE*: Este parámetro, con un tamaño que oscila entre 8 y 32K, determina el tamaño del *USER ÁREA*, es decir del área dedicada:

- en la versión 1.2 : a la ejecución de los programas NATURAL.
- en la versión 2.1 :

1) En tiempo de compilación: áreas de datos y tablas de funciones necesarias para la compilación.
2) En tiempo de ejecución: Áreas de datos locales (LDA), tabla de bucles área de stack del *CALLNAT*.

- *ESIZE* : Este parámetro establece el tamaño del *EXTENDED USER ÁREA*, pudiendo tomar los valores de 1 a 64K, y contiene:

1) Área de edición de fuentes.
2) Tabla de asignación de teclas de función.
3) Stacks de subrutinas.
4) Áreas de datos Globales.
5) Stack de comandos.

- *FSIZE* : Este parámetro establece el tamaño del *FILE BUFFER*, pudiendo tomar los valores de 2 a 64K, y está dedicado a contener:

1) DDMs durante la compilación.
2) Tabla de símbolos.

- *DSIZE* : Establece el tamaño del *DEBUG BUFFER*, pudiendo tomar los valores de 1 a 64K, y está dedicada a contener la salida de la utilidad ADA-LOG.

- *SORTSZE* : Establece el tamaño del *SORT BUFFER* y puede tomar los valores de 0 a 64K, dedicandose a ser el almacenamiento temporal para realizar el sort en procesos *On-line*.

CONCEPTO DE BUFFER POOL

Cuando un usuario entra en el sistema, se le dedica un buffer para cada uno de los tipos reseñados, tomándolos de un espacio de memoria dedicado a estos menesteres. Este Buffer Pool tiene su espacio de memoria troceado en páginas de 2 o 4Kb.

En principio, el tamaño dedicado al USIZE es de 3500 Bytes, pero a medida que se vayan invocando objetos, este tamaño se va incrementando en el número de bytes que ocupa cada uno de los objetos, liberándose el espacio dedicado a cada uno de ellos a medida que el proceso abandona cada módulo.

Cada vez que un módulo, del tipo que sea, invoca a un programa haciendo uso de la sentencia *FETCH* (esta regla no es valida si se ejecuta *FETCH RETURN*), se libera toda la memoria dedicada al *USIZE*, volviendo a tomar el valor de 3500 Bytes, más el tamaño del programa invocado.

UTILIDAD

La utilidad que acompaña a este artículo complementa la dada el pasado mes. Es una herramienta muy fácil de desarrollar, prácticamente los fuentes vienen distribuidos con el disco que acompaña a la revista, y sólo hay que sustituir el nombre de la view *SYSTEM-FUSER* por el nombre de la DDM que apunta al fichero *FUSER* del sistema, que en principio coincidirá con el de la utilidad.

Las ventajas son:

1) Lista cualquier fuente de cualquier librería, saltándose el control de *NATURAL-SECURITY*.
2) Permite buscar cadenas dentro de fuentes.
3) Con PF7 se relacionan los objetos referenciados en un objeto.
4) Su autor la califica como herramienta imprescindible en cualquier centro de desarrollo por su potencia, sencillez de uso, y sobre todo para funciones de mantenimiento.

El Fichero LISPGM.NAT, incluido en el disquete que acompaña a la revista, contiene el listado formateado de los distintos fuentes de la utilidad.

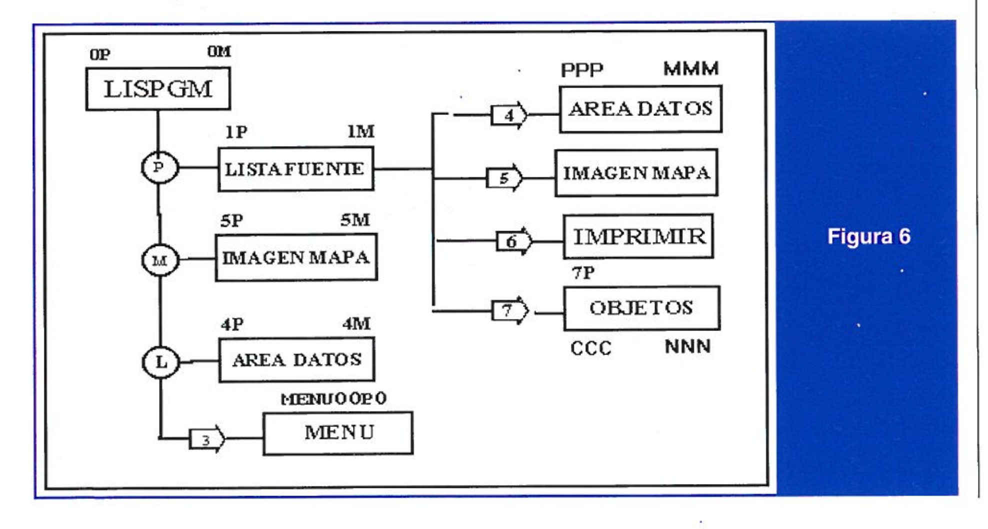
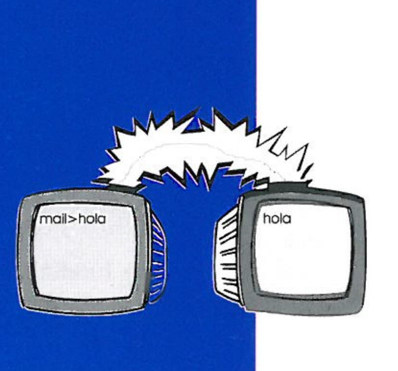


Figura 6

PROTOCOLOS TCP/IP

María Jesús Recio



En la actualidad se ha puesto tan de moda las conexiones a Internet, que el conocimiento de los protocolos TCP/IP se hace practicamente imprescindible; eso sin citar su utilización en redes locales. En esta entrega se desvelan ahora todos sus secretos para diferentes sistemas operativos.

En este artículo se va a hablar de TCP/IP, primero desde un punto de vista teórico, en el que se citarán características de TCP/IP, su localización dentro de los niveles, y a continuación desde un punto de vista práctico, relatando en ésta parte su forma de trabajo en diferentes sistemas operativos, asi como su configuración y puesta en marcha.

TCP/IP forma parte de la familia de protocolos Internet implementados por el DOD (*Departament Of Defense*) de los EE.UU., desarrollado a partir del año 1972. Para cada nivel del modelo se implementan distintos protocolos, como muestra la tabla 1.

Aunque el modo de manejo de estos protocolos es diferente para cada versión de sistema operativo, en todos los casos existe parte en común, y que es la propia definición de los protocolos TCP/IP, y otras definiciones tan importantes como son dirección IP, tipo de red, etc.

TCP/IP es el nombre común de una colección de más de 100 protocolos que permiten conectar ordenadores y redes. El nombre TCP/IP proviene de los dos protocolos más importantes: TCP (*Transmission Control Protocol*) e IP (*Internet Protocol*). Como se observa en la tabla 1, el protcolo TCP trabaja a nivel de transporte, mientras que el protocolo IP trabaja a nivel de red; por tanto, TCP utiliza los servicios prestados por IP.

TCP/IP permite interconectar redes independientemente del hardware que las implemente, por tanto va a abstraerse de la arquitectura de las máquinas que quieran conectarse: si a nivel físico se conectan, TCP/IP se encarga del resto. Recuérdese que el modelo TCP/IP no entra a considerar oficialmente el medio hardware como componenete específico en su modelo.

Una red TCP/IP transfiere datos mediante el ensamblaje de bloques de datos en paquetes. Cada paquete contiene: una primera parte, la cabecera que contiene información de control, y una segunda parte que contiene los datos.

PROTOCOLO IP

Se trata de un protocolo a nivel de red, cuyas principales características son:

- Ofrece un servicio no orientado a la conexión. Esto significa que cada trama en la que ha sido dividido un paquete es tratado por independiente: las tramas que componen un paquete pueden ser enviadas por caminos distintos e incluso llegar desordenadas.
- Ofrece un servicio no muy fiable, porque a veces los paquetes se pierden, duplican o estropean, y este nivel no informa de ello.
- Es en este nivel en el que se realiza la conmutación de paquetes, en el caso en el que sea necesario (no para redes locales).

DIRECCIONAMIENTO IP

Cada máquina con TCP/IP tiene asociado un número de 32 bits, al que se llama dirección IP, y que está dividido en dos partes:

1º Una parte que identifica la dirección de la red (NETID). Esta parte es

TABLA 1

CAPA	PROTOCOLO
Aplicación	Telnet, FTP, etc.
Transporte	TCP,UDP
Red	IP, ICMP
Enlace	ARP,RARP, etc.

Implementación Internet de la torre de niveles.

asignada por el *Network Information Center* (NIC), de la Red de Datos de Defensa (DDN), gobernada por *Network Solutions* en Cantilly, Virginia. Evidentemente, si nuestra red no va a contectarse con otras redes, no es necesario solicitar a este organismo una dirección. El número de bits que ocupa esta parte depende del tamaño de la red y puede ser 8, 16 o 24.

Para el NIC es esencial únicamente que las redes conectadas a Internet tengan y utilicen direcciones IP asignadas oficialmente. En una red de área local de una compañía se podría muy bien adoptar, como identificador de la red, un número cualquiera.

2º Otra parte que identifica la dirección de la máquina dentro de la red (HOSTID). Mientras que la dirección de la red debe ser asignada por el NIC, las direcciones de los hosts son asignadas por el administrador de la red. La primera dirección que se debe utilizar es la dirección 1.

Una dirección se representa por cuatro valores decimales, para que sea más facil su escritura y memorización. Por ejemplo, la siguiente direccion:
10000000.00001010.00000010.0001 1110
se representa por el valor:
128.10.2.30

MÁSCARA DE SUBRED

Cuando una red aparece segmentada (dividida en subredes), se debe utilizar un dispositivo que interconecte los segmentos. Si dicho dispositivo realiza funciones de filtrado, para que de este modo el rendimiento de la red sea mejor (cada subred trabaja de forma independiente, sin tener que ponerse de acuerdo con las otras subredes para transmitir), es necesario indentificar de algún modo cada uno de los segmentos. Si todos los segmentos tienen la misma dirección IP (en cuanto a la parte que identifica la red se refiere), se hace necesaria la existencia de algún mecanismo que diferencia los segmentos. Este mecanismo es la máscara de subred.

A cada dirección IP de red, es decir, a cada red física, se le asocia una máscara, que tambien es de 32 bits. La máscara sirve para dividir la parte de la dirección IP destinada a identificar el host en dos partes: la primera identificará ahora el segmento y la segunda el host dentro de este segmento. En esta máscara, los bits a 1 significan que el bit correspondiente de la dirección IP será tratado como bit parte de la dirección de subred, mientras que los bits a 0 indican que los bits correspondientes de la dirección IP seran interpretados como identificadores del host. Por ejemplo, la siguiente máscara, asociada a una red de tipo C (3 bytes para identificar la red y uno para el host dentro de la red), con dirección IP: 128.100.100:

De esta forma, con una mínima dirección de red se puede direccionar a muchas subredes. La combinación de ceros y unos puede ser cualquiera; no es necesario que aparezcan primero todos los unos, y despues los ceros, o viceversa, pueden mezclarse como se quiera.

Para identificar el host destinatario de la información transmitida, primero se localiza la red a la que pertenece, despues la subred (gracias a la máscara de subred), y por último el host final.

Existen una máscaras predeterminadas para cada tipo de red, y que se utilizan cuando estas no estan segmentadas (ver tabla 3).

TABLA 2

REDES TIPO A:
1...........................7
8...........................32

dirección de la red 0.......	identificador de la máquina

REDES TIPO B:
1...........................16
17...........................32

dirección de la red 10....	identificador de la máquina

REDES TIPO C:
0...........................23
24...........................31

dirección de la red 110....	identificador de la máquina

Sintaxis de las direcciones IP, dependientes del tipo de redes.

CLASES DE REDES

Ya se ha dicho que a la hora de dar una dirección de red es necesario conocer el tipo de nuestra red. El tipo (clase) depende del número de máquinas que formen la red. Atendiendo a esto se pueden encontrar redes de tres clases:

- Redes de clase A: Se trata de las redes de mayor tamaño, redes que tengan más de 2^{16} (65.536) hosts.
  El espacio reservado para la dirección de red en estas redes es más pequeño por dos motivos: porque existen menos redes de este tipo y porque al tener más hosts se necesita dejar más espacio para direccionarlos.
  La parte que identifica la red consta de:
  - Un cero (0)
  - 7 bits más.
  Por lo que se podrán direccionar 2^7 redes, que hace un total de 128 redes diferentes, con direcciones que van teóricamente desde la 0 a la 127. Cada una de estas redes podrá tener 2^{24}=2.097.152 posibles hosts. La dirección 127 NO se utiliza, por lo que en realidad la dirección oscilará entre 0 y 126.
- Redes de clase B: Son redes de tamaño mediano, que tengan entre 2^8

TABLA 3

TIPO DE RED	DIRECCION DE RED	MASCARA PREDETERMINADA
Clase A	XXX	255.000.000.000
Clase B	XXX.XXX	255.255.000.000
Clase C	XXX.XXX.XXX	255.255.255.000

Máscaras predeterminadas para cada tipo de red.

Figura 1: Estructura de un datagrama IP.

```
0........4..........8.......................16.......19..........24..........................................31
VERS | CALG | TIPO DE SERVICIO |            LONGITUD  TOTAL
           IDENTIFICADOR          | FLAGS |          FRAGMENTACION
TIEMPO DE VIDA | PROTOCOLO         |          CHECKSUM CABECERA
                     DIRECCION IP DE ORIGEN
                     DIRECCION IP DE DESTINO
            OPCIONES IP (SI EXISTEN)             |        PADDING
                              DATOS
```

(256) y 2^{16} (65536) hosts. La parte que identifica a la red consta de:

- La secuencia uno-cero (10).
- 14 bits con cualquier valor.

Por tanto, el rango de valores para el primer byte de los dos asignados a la red en este tipo de redes es: 128-191 (16.384 redes de clase B). Si se echan cuentas se puede calcular que estas redes pueden tener 2^{16}= 65.536 hosts.

- Redes de clase C: Se trata de las redes de menor tamaño, aquellas que tienen hasta 2^8 hosts (256). En este caso, se reservan 3 bits para identificar la red. La parte de la dirección que identifica la red consta de:
  - La secuencia uno-uno-cero (110).
  - 21 bits con cualquier valor.

  Por tanto, el rango de valores del primer byte correspondiente a la red estará entre 192-223. El último byte de la dirección identifica al host dentro de la red, por lo que se puede deducir que el número máximo de hosts para este tipo de redes es de 2^8=256 hosts.

CONVENCIONES DE DIRECCIONES ESPECIALES

Existen algunas direcciones que no se asignan como direcciones IP, sino que tienen asignado un significado especial. Estas combinaciones son:

- Dirección de red (todos unos o todos ceros): Como por ejemplo la dirección 192.000.000.000 o bien 192.255.255.255. Se llaman direcciones de multidifusión dirigida a la red. Estas dos combinaciones de dirección permiten direccionar a todas las máquinas dentro de la red especificada (*broadcast*). Ambas significan lo mismo, unicamente que la primera de ellas (todos a unos), es la utilizada por las últimas versiones de UNIX, mientras que la segunda (todos a ceros), es la utilizada en el convenio BSD. De hecho, todas las máquinas deberían definir las dos direcciones, para no tener problemas.
- Todos unos (255.255.255.255): Se llama dirección de multidifusión local, y permite direccionar a todos los hosts de la red local, independientemente de su dirección de red IP.
- *127.* cualquier combinación (habitualmente todos ceros seguidos de un 1, es decir, 127.000.000.001): se llama dirección de *loopback*. Se trata de una dirección local a la máquina. Está diseñada para realizar pruebas y comunicaciones entre procesos dentro de una misma máquina. Si un programa envia un mensaje a esta dirección, TCP/IP le devolverá los datos sin enviar nada a la red aunque se comporta como si lo hubiera hecho.

Una dirección Internet NO identifica a un host, sino a una conexión a una red. Esto quiere decir que si se tiene una estación puente conectada por tanto a dos redes, esta estación tendrá dos direcciones IP ya que, si no fuera asi, ¿qué dirección Internet se le daría a esta estación? (tiene dos posibles direcciones, una por cada red a la que está conectada).

EL DATAGRAMA IP

La red Internet denomina a su unidad básica de transferencia DATAGRAMA INTERNET, conocido tambien como datagrama IP o simplemente como datagrama (ver figura 1). Un datagrama está dividido en dos partes:

1- Encabezamiento: Contiene muchos campos, entre los que se pueden destacar dirección IP de origen, dirección IP de destino, tipo de trama.

2- Datos: contiene la información que se quiere enviar.

Como se puede observar en la figura 1, el datagrama IP está dividido en muchos campos de control, que manejan información necesaria para la correcta transmisión.

Para que el datagrama se transmita de un nodo a otro de la red, deberá ser transportado en un paquete de la red física, llamada trama. La idea de transportar un datagrama en una trama de red se llama encapsulamiento (ver figura 3).

Se debe recordar que para la red física el datagrama IP es como cualquier mensaje intercambiado entre dos ordenadores, sin que reconozca ni el formato del datagrama, ni la dirección de destino IP. En el caso ideal, todo datagrama IP cabría en una sola trama de red. Pero como el datagrama puede atravesar en su camino diferentes tipos de redes físicas, no existe una longitud máxima de datagrama que se ajuste a todas las redes, por lo que en la mayoría de los casos hay que dividir el datagrama en tramas físicas.

A la longitud máxima de transferencia de datos por trama de una red física se le conoce con el nombre de UNIDAD DE TRANSFERENCIA MÁXIMA (MTU). Cuando un datagrama se envía por una red con MTU menor que su longitud, entonces el datagrama se divide en partes llamadas fragmentos. A este proceso se le conoce como fragmentación (ver figura 4).

Con TCP/IP, una vez que el datagrama ha sido fragmentado, cada fragmento viaja por la red separado del resto hasta el destino final, donde deben ser unidos de nuevo. Esta fiolosofía de trabajo tiene algunas desventajas:

- Si en el camino hasta llegar al destino se encuentra una red con un MTU mayor que la red desde la que partió el datagrama, no se aprovechará la capacidad de esta.

Figura 2: Estructura de un segmento TCP.

```
0.........4.........................10..............16.........................24........................31
      puerto origen                           |                  puerto destino
                             número de secuencia
                             acknowledgement number
LH        | reservado        | code bits |                 window
            checksum                      |
urgent pointer
                      options (if any)                       |     paddign
                                  datos
                                   ...
```



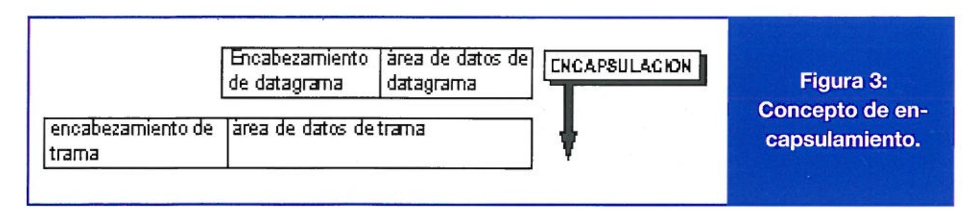


Figura 3: Concepto de encapsulamiento.

- Si se pierde algún fragmento se debe retransmitir el datagrama completo, ya que la máquina destino, cuando recibe el primer fragmento de un datagrama, inicializa un temporizador. Si el temporizador rebasa un valor máximo antes de que lleguen todos los fragmentos, se descartan los fragmentos recibidos sin procesar el datagrama.

RELACIÓN ENTRE DIRECCIONES IP Y DIRECCIONES FÍSICAS

Se ha visto cómo una trama IP es encapsulada en una trama física para su transporte por la red. Esto significa que en algún punto se tiene que relacionar la dirección IP suministrada con una dirección física.

Si se recuerda la jerarquía de niveles utilizada por Internert, se puede observar que por debajo del protocolo IP (situado en el nivel de red) se encuentra el nivel de enlace, en el que se hallan protocolos como ARP (*Address Resolution Protocol*) o RARP (*Reverse Address Resolution Protocol*) destinados a resolver problemas relacionados con las direcciones.

La forma en que el protocolo resuelve los problemas de las direcciones se puede describir como sigue: cuando desconoce la dirección física de una máquina envía un paquete por multidifusión, es decir, a todos los segmentos conectados, con lo que él conoce su dirección IP. Al llegar a su destino, éste devuelve un paquete con su dirección física. La relación entre estas dos direcciones se apunta en una tabla de encaminamiento para futuras conexiones. El protocolo RARP realiza la conversión inversa de direcciones.

PROTOCOLO TCP

A este nivel, los usuarios pueden imaginar que existe un tubo directo de comunicación entre el ordenador origen y el destino. Las principales características de este protocolo son:

- Se trata de un protocolo orientado a la conexión (trata un paquete como una entidad).
- Orientado al flujo: el servicio TCP envía al receptor los datos en el mismo orden en el que fueron enviados.
- Conexión con circuito virtual: no existe conexión física dedicada. Sin embargo el protocolo hace creer a la aplicación que sí exite ésta conexión dedicada.

La unidad de transferencia TCP entre dos máquinas se llama segmento. Cada segmento se divide en dos partes: encabezamiento y datos. La estructura de cada segmento aparece descrita en la figura 2.

FICHERO HOSTS

El fichero hosts es uno de los ficheros de configuración más importantes para TCP/IP. En él se establece la relación existente entre la direccion IP de una máquina y su nombre.

A la hora de utilizar servicios de TCP (ftp, telnet, rlogin, talk), al usuario le resulta mucho más cómodo identificar la máquina con la que quiere conectarse a través de un nombre, en lugar de hacerlo a través de su dirección IP. Debido a esto, es necesario establecer en algún sitio la relación direccion-nombre. Ese sitio es el fichero hosts.

El fichero hosts (habitualmente localizado en máquinas UNIX en el directorio /etc) contiene una linea por cada relación establecida. La sintaxis de una linea es como sigue:

dirección-IP   nombre   alias

Cuando una máquina quiere establecer comunicación con otra, a través de un comando TCP y utilizando su nombre, o a través de un alias, se utiliza este fichero para localizar su dirección IP. Por tanto, éste fichero debe contener todas las relaciones de todas las máquinas a las que en algún momento se quiera conectar a través del nombre. Si se intenta contactar con una máquina a través del nombre y sin que aparezca definida en este fichero, el sistema producirá un error.

El fichero hosts existe tanto en la versión de TCP/IP para UNIX como en la versión para DOS.

El nombre de una máquina, al que se denomina dominio, está dividido en subdominios, en los que se identifica elementos como nombre de máquina en sí, nombre del organismo/empresa en el que está localizada la máquina y país en el que está ubicada, o bien dedicación del organismo/empresa. Lo habitual es que el nombre de una máquina tenga 3 partes, es decir, que el dominio de una máquina tenga 3 subdominios, de los cuales el de más a la derecha se llama subdominio de primer nivel, y es el más genérico de todos. Un ejemplo de nombre, podría ser el siguiente: afrodita.cefat.es

Aunque, si estamos en el entorno de una red local, sin conexión al exterior, el nombre de las máquinas que la integran podría ser unicamente el nombre de la máquina. Siguiendo el ejemplo anterior, el nombre de una máquina seria: afrodita

LOCALIZACIÓN DE AVERÍAS EN UNA RED TCP/IP.

Las redes, algunas veces, se empeñan en no funcionar. Cuando ésto ocurre, el administrador de la red debe primero localizar el fallo, y después arreglarlo. Para localizar el fallo, TCP/IP incorpora algunas herramientas, como son: *ping*, *rup*, *traceroute*, *netstat*, *ifconfig*, etc. No todas las herramientas estan disponibles en todas las versiones de TCP/IP.

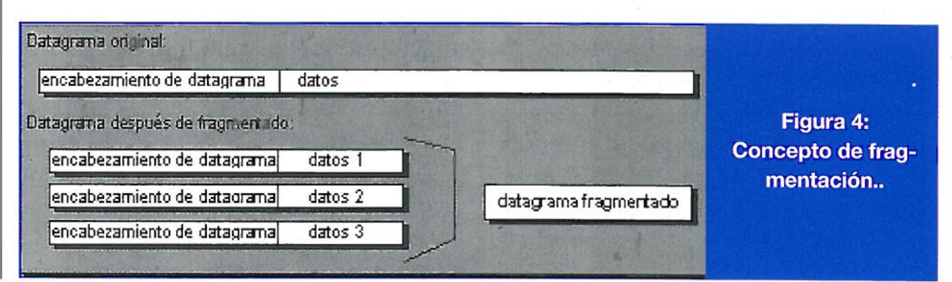


Figura 4: Concepto de fragmentación..

TABLA 4	
DEMONIO	**SERVICIO**
ftpd	FTP (transferencia de ficheros)
rlogind	rlogin (conexión remota)
rshd	rsh (shell remota)
rexecd	rexec
telnetd	telnet
talkd	talk
inetd	echo, discard, chargen, daytime.
rarpd	solo si existe el protocolo RARP, para darle soporte.

Demonios y servicios de TCP.

El comando *ping* permite detectar averías físicas de la red, pues comprueba la conexión de una máquina a nivel de interface de red. Es decir, sólo comprueba si es posible "llegar" hasta una máquina, independientemente de si ésta está capacitada para ofrecer servicios TCP. Si *ping* falla, es decir, si nos dice que no detecta la máquina, y se está intentando realizar una comunicación a través de un encaminador, se puede utilizar la herramienta *traceroute*, que da como resultado de su ejecución la ruta que sigue la información desde que sale de la máquina origen hasta que llega (o falla) al destino, además de mostrar el tiempo invertido.

El comando *rup* ofrece un nivel más sobe *ping*. Dice si una máquina esta activa o no, es decir, si responde al tráfico de la red.

Netstat e *ifconfig* son comandos más avanzados, que ofrecen información sobre los encaminadores, tráfico, tipos de servicios que el sistema está escuchando y estádisticas sobre el interfaz de red.

TCP/IP PARA UNIX.

La gestión de redes bajo Unix está enteramente integrada en el propio núcleo, de forma que existen unos módulos capaces de gestionar cada capa de red (según el modelo OSI).

TCP/IP ofrece varios servicios: *telnet*, *ftp*, *rlogin*, *mail*, etc. Cada uno de estos servicios debería estar a la escucha del medio para saber cuándo se solicita. Sin embargo, esto supondría crear muchos procesos que, en prinicpio, se dedicarán a escuhar el medio hasta que les llegue una petición y entonces servirla. Como este método es inviable, pues supondría la creación de muchos procesos, en la mayoría de los casos innecesarios, Unix implementa éste otro mecanismo:

Crea un proceso demonio inetd (*internet daemon*), que escuha en el medio todos aquellos paquetes TCP/IP (independientemente del servicio) cuya dirección IP de destino coincida con la máquina actual. Cada paquete TCP tiene un campo llamado puerto destino, que identifica el servicio que se solicita en la máquina a la que va dirigido dicho paquete. Este campo es utilizado por el demonio inetd para identificar el servicio que se solicita y así llamar al demonio correspondiente. Ademas, el propio demonio inetd ofrece servicios como *echo*, *discard*, *chargen*, *daytime*.

Los demonios implicados y los servicios que cada uno de ellos gestiona aparecen reflejados en la tabla 4.

Para el correcto funcionamiento de todos estos procesos, se crean unos ficheros de configuración, como son: hosts, services, networks, netmasks, rhosts, protocols, etc.

- Fichero *services*.

  A nivel TCP, la información se direcciona no sólo con la dirección IP que identifica la máquina sino, además, con el tipo de servicio que se solicita. Este último valor aparece en el campo puerto destino, que puede observarse en el segmento TCP (figura 2). Igual que la obtención de la dirección IP de una máquina a partir del nombre se soluciona con el fichero hosts, la obtención del nombre del servicio solicitado a traves de su número (puerto) se resuelve gracias al fichero services. Este fichero suele estar localizado en el directorio */etc*.

  Este fichero contiene una línea por cada relación "puerto-nombre de servicio, protocolo base del servicio". Evidentemente estas relaciones deben ser estándar para que no exista confusión entre las máquinas comunicantes. Un ejemplo de este fichero aparece reflejado en el listado 1.
- Fichero *networks*.

  Este fichero permite identificar una red entera, en lugar de con su dirección IP, a través de un nombre, al igual que el fichero hosts lo hace para las estaciones. Cada dirección de red puede estar así asociada a un nombre lógico más explicíto. El modo de funcionamiento cuando se tiene este fichero es exactamente el mismo que el descrito para el fichero hosts extrapolándolo a redes completas (en lugar de estaciones).

  Debe señalarse que la existencia de este fichero no es obligatoria, y por tanto que TCP/IP funciona correctamente sin él (evidentemente sin poder utilizar las prestaciones que ofrece). Un fichero networks puede ser como el que aparece en el listado 2.
- Fichero *protocols*.

  El fichero protocols contiene todos los protocolos instalados en la máquina. Contiene mínima información acerca de los protocolos utilizados por el DARPA Internet. Este fichero se crea cuando se instala TCP/IP, y no necesita ningún mantenimiento por parte del administrador de la red. La estructura de este fichero de nuevo vuelve a ser la misma: líneas de definición de protocolos con 2 partes: nombre oficial del protocolo, número de protocolo.
- Fichero *hosts.equiv*.

  El fichero hosts.equiv contiene el nombre de las máquinas remotas que pueden utilizar servicios como *telnet*, *rlogin*, *lp* y *rsh* en la máquina

LISTADO 1

```
# UNIX specific services

echo                  7/TCP
discard  9/TCP
discard  9/udp
systat                11/TCP
daytime  13/TCP
daytime  13/udp
ftp                   21/TCP
telnet                23/TCP
smtp                  25/TCP        #mail
time                  37/TCP
time                  37/udp
finger                79/TCP
whois                 53/TCP
pop                   109/TCP
snmp                  161/udp
src                   200/udp
auth                  113/TCP
exec                  512/TCP
login                 513/TCP
who                   513/udp  #whod
shell                 514/TCP
syslog                514/udp
route                 520/udp
talk                  517/udp
```

Fichero services

en la que se encuentra definido, sin necesidad de introducir ni *login* ni *password*. Suele estar localizado en el directorio */etc*.

Evidentemente, para que las cosas funcionen correctamente, las máquinas contenidas en este fichero deben tener nombres de cuenta (usuarios definidos) iguales a las de la máquina que lo contiene. Cuando se ejecuta alguno de los comandos remotos descritos, para establecer conexión con una máquina remota (máquina con la que se quiere establecer la conexión), y el nombre de la máquina local (máquina desde la que se hace la peticion) está contenido en el fichero *hosts.equiv* de la máquina remota, entonces se entra directamente a la cuenta cuyo *login* coincida con el *login* que se tiene en la máquina local.

Su sintáxis obiliga a escribir el nombre de cada máquina en una nueva linea. Por tanto, contendrá tantas líneas como máquinas se quieran definir en él. Un ejemplo del contenido de este fichero puede ser el siguiente:

```
#Máquinas equivalentes:
rs6000
afrodita
```

- Fichero *netmasks*.
  Mediante éste fichero, ubicado habitualmente en el directorio */etc*, se pueden definir máscaras de dirección para crear subredes. Contiene una linea para cada subred definida. Cada línea tiene la dirección de red y a continuación la máscara.
- Fichero *rhosts*.
  A diferencia del resto de los ficheros descritos, éste fichero no es único en la máquina, y se localiza en los directorios */home* de cada usuario. El dueño de cada uno de estos ficheros debe coincidir con el dueño del usuario al que pertenece el directorio en el que se guarda, y además debería tener únicamente permiso de lectura y escritura para él para evitar que intrusos pudieran modificarlos. Permite a cada usuario definir quién va a poder utilizar comandos remotos desde otra máquina y entrada a su cuenta.

La estructura de este fichero es la siguiente. Cada linea define a un usuario que puede conectarse a la cuenta, junto al nombre de la máquina a la que pertenece dicho usuario. Por ejemplo, la línea siguiente en el fichero *rhosts*:

```
venus geo
```

quiere decir que el usuario geo de la máquina venus tiene entrada a la cuenta. En este fichero se pueden encontrar lineas que únicamente contengan un nombre de máquina. En este caso el sistema asume que el *login* deberá coincidir con el *login* que se tiene en la máquina remota. De esta forma cada usuario podría permitir la entrada de otros usuarios en la máquina. Debe tenerse cuidado con esto, pues podrían entrar usuarios no deseados.

TCP/IP PARA DOS

A diferencia del TCP/IP para Unix, en MS-DOS este paquete no está incluido, por lo que es necesario comprarlo como software adicional. Existen diferentes versiones de TCP/IP para MS-DOS distribuidos por distintas casas comerciales. La diferencia entre ellas estriba en los "comandos" que ofrece al usuario y su sintáxis, sin hablar indudablemente de su funcionamiento interno y su gestión a nivel de sistema. Además existen versiones de TCP/IP para DOS y para entornos Windows. En estas últimas los comandos siguen la filosofía de las aplicaciones Windows, ejecutándose por tanto con un interface gráfico y teóricamente fácil de usar.

Durante el proceso de instalación de TCP/IP para DOS se hacen algunas preguntas, como son: dirección IP, nombre del host y tarjeta de red instalada, ademas de la submáscara de red. Es conveniente, cuando se instala, tener a mano el driver de la tarjeta, porque a veces el software de instalación no la reconece.

Una vez instalado el software es necesario configurar el fichero hosts con la relación nombre de máquina/dirección IP de todas las máquinas con las que en algún momento se quiera establecer conexión. Además, para hacer funcionar cada uno de los servicios ofrecidos por TCP/IP, es necesario configurar el apartado correspondiente del fichero pcTCP.ini. Por ejemplo, si se quiere utilizar los servicios de mail, es necesario especificar una serie de datos en este fichero, al igual que si se desea poner clave de acceso a la máquina cuando ésta esté ejecutando un servidor de ftp.

Teniendo en cuenta que DOS es monotarea y monousuario, no se va a disponer de métodos de seguridad de acceso a la máquina, y tampoco de servicios múltiples. Es decir, cuando se ejecuta un servidor de algún servicio TCP, por ejemplo, un servidor de ftp, la máquina DOS que lo ejecuta se queda en modo dedicado (al igual que cuando se arranca cualquier otra aplicación DOS), hasta que se aborte la ejecución del servidor.

LISTADO 2

```
# Redes Internet

loopback     127
inge         192.0.200   # Red de desarrollo
comerciales  192.9.100   # Departamento co-
mercial
```

Fichero networks

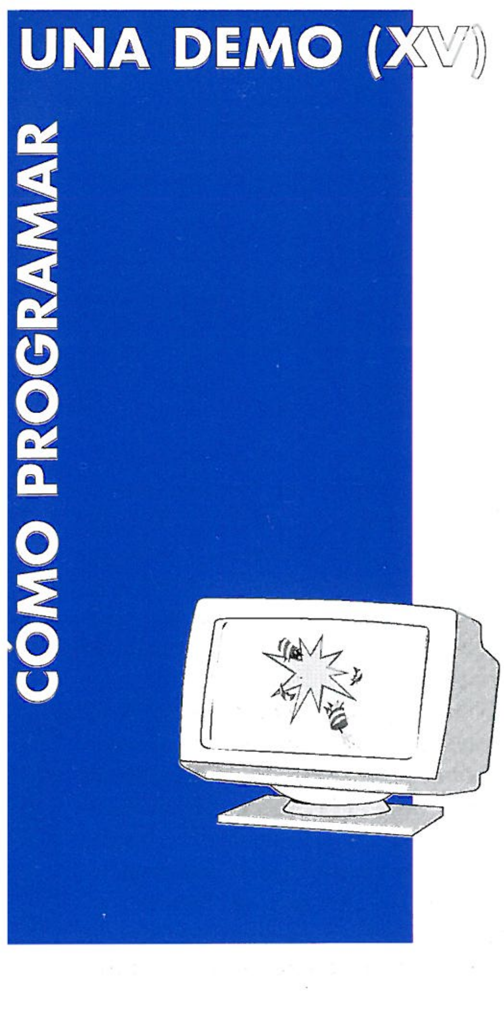
TCP/IP EN WINDOWS 95

Windows95 ofrece su propio software de TCP/IP incluido en el sistema. Sin embargo, las utilidades de TCP/IP que Windows95 ofrece son muy pocas, por lo que se ve limitada la potencia de estos protocolos.

Los comandos ofrecidos por esta reducida versión de TCP/IP son básicamente *arp*, *ftp*, *nbstat*, *netstat*, *ping*, *route*, *tracert*, y *telnet*.

Sobre Windows95 se puede instalar TCP/IP además de otras pilas de protocolos, como son IPX/SPX, el propio Microsoft Network, LANtastic, SunSoft, etc, y funcionar con todas de forma paralela.

Unicamente la versión Plus! de Windows95 ofrece protocolos TCP/IP. Para instalarlos hay que ejecutar la aplicación red que se encuentra dentro del panel de control. Al igual que en DOS, se solicitará la dirección IP, el nombre del host, y la submáscara de red. En esta versión de TCP/IP se ofrecen algunas facilidades para completar el fichero hosts.



EL RELLENADO SÓLIDO

Pedro Antón

Este mes se retoma el tema de las 3D con uno de los objetivos de este curso: aprender cómo hacer un rellenado sólido. En esta nueva entrega se explicará cómo crear una rutina de rellenado de triángulos aplicada al ya popular donut.

Aquellos lectores interesados en las 3D estarán esperando ansiosamente alguna forma de rellenar el donut que nos acompaña desde el ultimo numero. Quizás en el próximo numero se cambie el donut por el pato. Dos figuras muy explotadas en el mundo de las demos. Pues bien, esta vez se explicará como hacer un rellenado sólido de forma rápida y a su vez clara, algo que, por otra parte, es objetivo primordial en este curso.

LA TERCERA DIMENSIÓN

Hasta ahora, debe haber quedado claro cómo definir un objeto en tres dimensiones por sus puntos y sus caras. Si en el anterior artículo se dijo que un objeto físico no está formado únicamente por puntos, en éste se puede decir que tampoco lo está únicamente por líneas, ¿no?. Los objetos físicos generalmente son solidos. Veamos cómo generar esa simulación.

EL RELLENADO SÓLIDO

Como se ha visto, la unidad mínima de superficie es un triángulo, y combinandolos adecuadamente se puede generar cualquier superficie, sin limitación en su número de lados.

En primer lugar se determinarán los segmentos que conforman la línea, como se puede apreciar en la figura 1.

Para calcular estos valores es necesario conocer dos puntos de la recta, generalmente los extremos. El lector puede imaginar la necesidad de recurrir, una vez más, a las matemáticas.

La ecuación de una recta es $y = mx + b$, donde m es la pendiente de la recta, o dicho de otro modo, las variaciones de la coordenada x respecto de la coordenada y. El valor de b representa el punto de corte de la recta con el eje y, es decir, el valor que toma la recta cuando x toma valor cero. Esto se aprecia claramente en la figura 2.

Pero en realidad no es necesario conocer el valor de y cuando x vale cero. De esta forma, la fórmula quedará como se muestra en la fórmula 1.

Es sabido de todos cómo funcionan estas ecuaciones. Como se ha visto en las matemáticas mas elementales, con esta fórmula se puede obtener cualquier valor de y dando valores a la x para conseguir todos los valores de x y de y donde se va a pintar.

Cuando se trabaja con accesos lineales a la memoria de vídeo, como en el caso del modo 13h (320x200), resulta mucho mas rápido escribir líneas hori-

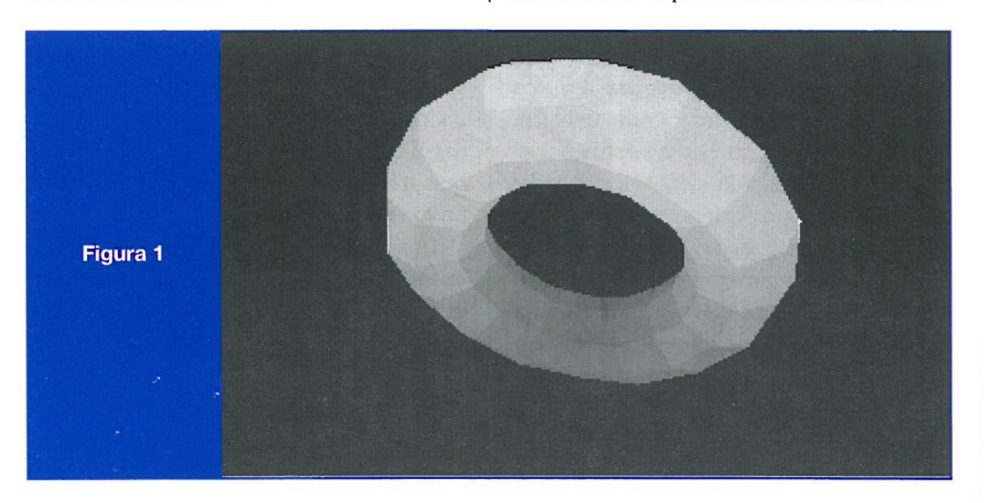


Figura 1

LISTADO 1

```
FOR Cnt:=Y1 to Y2 do
  Begin
    X:=m((Y1+Cnt)-Y1)+X1;
  End;

FOR Cnt:= Y1 to Y2 do
  Begin
    X:=m*Cnt+X1;
  End;
```

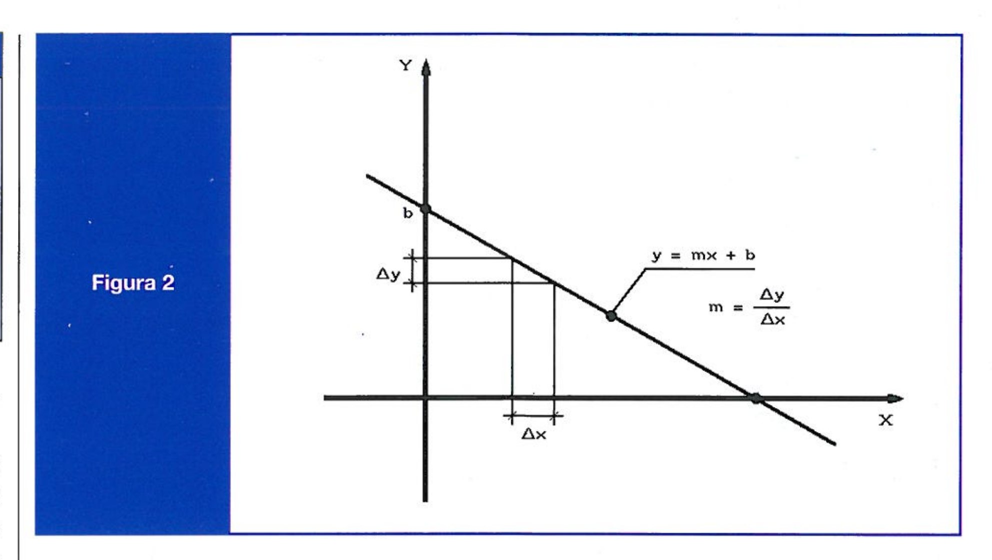


Figura 2

zontales. Luego será necesario conocer el valor de x para una y dada.

Al comenzar a calcular los datos necesarios para rellenar el triángulo, se debe calcular la pendiente de las aristas (ver fórmula 1). Supongamos una arista definida por los puntos *(X1,Y1)* y *(X2,Y2)*, los datos que es necesario conocer de una arista son todos los valores de x para los valores de y, comprendidos entre *Y1* e *Y2*, de esta forma se obtendrá dónde se empezará y donde se terminará de pintar líneas horizontales a la hora de rellenar el polígono.

Ya estamos listos para calcular la ecuación y resolver el problema planteado, pero esto será un poco lento. Ahora bien, si se piensa en cómo implementar dicho algoritmo, rápidamente se ve cómo al empezar a calcular valores desde *Y1* hasta *Y2* e implementar la ecuación en el código, el valor de *Y-Y1* se ve modificado e igual al contador que se emplee en la implementación, como se aprecia en el Listado 1.

Lo mismo ocurre con valores consecutivos de x. Sean dos valores de x, $x=n$ y $x=n+1$, aplicándolos a la formula, se puede observar:

$$(m(n+1)+X1)-(m(n)+X1)= mn+m+X1-mn-X1 = m$$

Un objeto físico no está formado únicamente por puntos ni por líneas

La conclusión que se obtiene de este desarrollo es que el valor de cada término está basado en el anterior. Así pues, comenzando por el primer valor de x (conocido e igual a *X1*) y añadiendo la cantidad m al valor de x del anterior valor de y, se determina el actual valor de y.

Surge ahora un nuevo problema, la pendiente es el resultado de una división, y por supuesto, no será suficiente su parte real. Será necesario almacenar dicho valor como un número real. Ante esto, existen dos posibilidades: trabajar con el coprocesador matemático, o usar punto fijo. En el código que acompaña a este artículo se opta por la segunda opción, como se puede apreciar en el listado 2.

Esta es la solución óptima para calcular los incrementos de las aristas a gran velocidad. Ahora bien, para calcular los incrementos de todas las aristas de un triángulo, se seguirán los siguientes pasos:

1- Definir dos listas de incrementos: una para la derecha, y otra para la izquierda.
2- Calcular los incrementos de la arista mas larga y almacenarla en la lista de la izquierda.
3- Convertir las otras dos aristas y almacenar los incrementos en la lista de la derecha.

Y para rellenar el triángulo de un sólo color, a estos pasos se añadirá un cuarto:
4- Dibujar líneas horizontales desde el valor calculado en la lista de la izquierda hasta el valor calculado en la arista de la derecha.

LISTADO 2

```
Incline:=(X2-X1) shl 6;
If (Y2-Y1)<>0 then Incline:=Incline div (Y2-Y1)
        else Incline:=0;
FixX:=X1 shl 6;
For CntY:=Y1 to Y2 do
  Begin
    Edge1[CntY]:=FixX shr 6;
    FixX:=FixX+Incline;
  End;
```

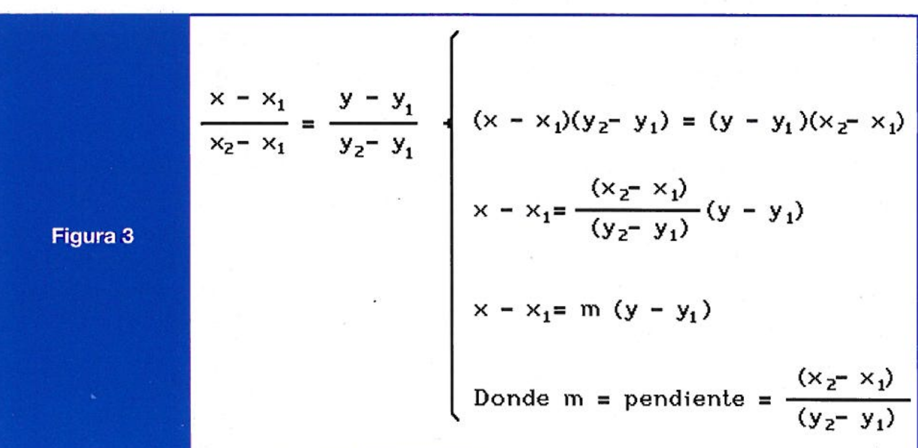


Figura 3

LISTADO 3

```
;
; HORIZONTAL LINE
; In : (X1,X2) = (bx,cx)
;      di    = y
;      al    = Color.
;      es    = Segmento destino.
HLine PROC
mov  ax,SegDes
mov  es,ax
mov  al,Col
mov  ah,al
push ax
shl  eax,16
pop  ax
mov  di,CntY
mov  dx,di
xchg dh,dl
shl  di,6
add  di,dx
add  di,bx
sub  cx,bx
jz   @@NoLine
mov  dx,cx
shr  cx,2
rep  stosd
and  dx,3
jz   @@NoLine
mov  cx,dx
rep  stosb
@@NoLine:
ret
HLine ENDP
```

Este último paso debe hacerse de la forma más rápida posible. Además, debe estar integrado en la rutina de rellenado para ahorrar el mayor número de ciclos posibles.

Veamos entonces cómo dibujar líneas horizontales de la forma más rápida posible. Se conocer cuatro valores: la coordenada *x* inicial, la coordenada *x* final, la coordenada *y* y el color. El proceso a seguir es trivial, no obstante los pasos serán:

1- Determinar la dirección de memoria donde se comenzará a dibujar la línea.
2- Calcular la longitud de la línea.
3- Escribir tantos bytes como la longitud del color dado.

Una solución, bastante eficiente es la mostrada en el listado 3

EL *CLIPPING*

Recortar las imágenes si éstas se salen de la pantalla es algo absolutamente necesario en una buena rutina de rellenado.

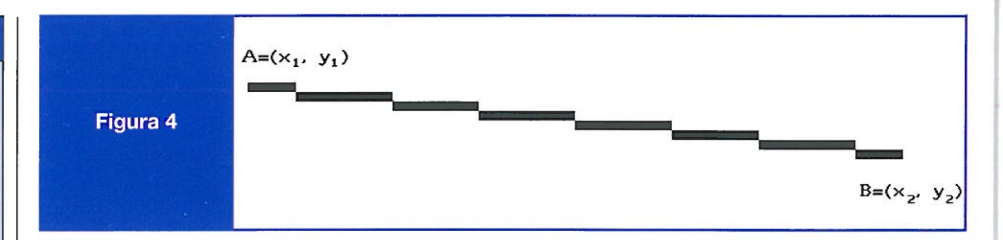
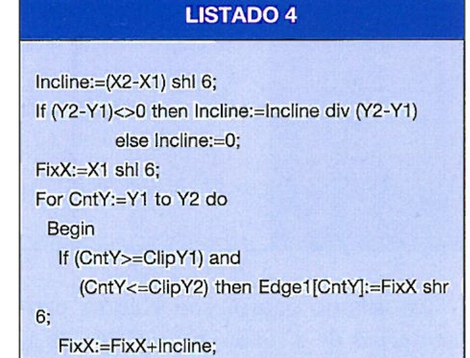


Figura 4



LISTADO 4

```
Incline:=(X2-X1) shl 6;
If (Y2-Y1)<>0 then Incline:=Incline div (Y2-Y1)
        else Incline:=0;
FixX:=X1 shl 6;
For CntY:=Y1 to Y2 do
 Begin
  If (CntY>=ClipY1) and
     (CntY<=ClipY2) then Edge1[CntY]:=FixX shr
6;
  FixX:=FixX+Incline;
 End;
```

En modo 13h, cuando el triángulo se salga por los laterales de la pantalla, aparecerá en la parte contraria de ésta, siempre y cuando se trabaje en modo real. En modo protegido pueden derivarse desagradables consecuencias.

Recortar el polígono en *y* es sencillo. Basta con incluir un par de condiciones en la rutina de cálculo de incrementos (ver listado 4).

En este listado se observa cómo se incrementa el valor de x constantemente, pero sólo se almacenan en el array aquellos valores que se encuentran dentro de la pantalla.

Recortar el polígono en *x* es mucho más sencillo. Como lo que se pinta son líneas horizontales, se comprueba si la *x* es menor que cero, en cuyo caso se hace cero; y si es mayor que el ancho lógico de la pantalla (en este caso 320) se iguala a dicho ancho.

LA UNIDAD DEMOVGA

Se ha incluido en la unidad la rutina de rellenado sólido con *clipping*.

En el listado 5 se puede apreciar la nueva rutina incluida en la librería. El lector puede contrastar que la no inclusión de las instrucciones *sar* y *sal* en el compilador hace que el listado 5 y el código que acompaña a este artículo no se correspondan, aunque ambos sean técnicamente correctos. El motivo es presentar el listado 5 lo mas claro posible. Destacar que la diferencia entre las instrucciones *shr*, *shl* y *sar*, *sal* es únicamente el tipo de dato con el que operan. Mientras que las primeras lo hacen con operadores sin signo, las segundas operan con datos con signo.

Los pasos que sigue la rutina son:

1- Hacer que el punto más alto del triángulo sea el primero.
2- Poner la arista más larga del triángulo entre los puntos primero y segundo.
3- Calcular los incrementos de *x* de las tres aristas, recortados en *y*.
4- Recortar los límites de *y* del triángulo si es necesario.
5- Pintar todas las líneas horizontales dentro de los limites de *y*, comprobando si la línea a pintar es de izquierda a derecha o de derecha a izquierda.

El lector puede observar en este listado que el punto fijo empleado en esta rutina es de 6. No se emplea mayor definición por dos motivos. El primero de ellos es que no es necesaria una definición mayor para esta resolución. Y el segundo es que emplear otra resolución, como pueda ser 8.8, limitaría, si se usan enteros, los valores de *x* e *y*. Pero esto es algo que el lector podrá comprobar por sí mismo experimentando con el código que acompaña a este artículo.

Combinando triángulos adecuadamente se puede generar cualquier superficie, sin limitación en su número de lados

LISTADO 5

```
PROCEDURE FlatFill
(SegDes:Word;X1,Y1,X2,Y2,X3,Y3:Integer;
 Col:Byte;ClipX1,ClipY1,ClipX2,ClipY2:Integer);
Var
  Edge1   : Array [0..199] of Word;
  Edge2   : Array [0..199] of Word;
  SwapVar : Integer;
  FixX    : Integer;
  Incline : Integer;
  CntY    : Integer;
  XH1,XH2 : Integer;
Begin
{ X1 to upper }
  If Y1>Y2 then Begin { Changes (X1,Y1) (X2,Y2) }
             SwapVar:=X1;
             X1:=X2;
             X2:=SwapVar;
             SwapVar:=Y1;
             Y1:=Y2;
             Y2:=SwapVar;
           End;
  If Y1>Y3 then Begin { Changes (X1,Y1) (X3,Y3) }
             SwapVar:=X1;
             X1:=X3;
             X3:=SwapVar;
             SwapVar:=Y1;
             Y1:=Y3;
             Y3:=SwapVar;
           End;
{ 1-2 line will be the largest ;)    }
  If (Y2-Y1)<(Y3-Y1) then Begin { Changes (X2,Y2)
(X3,Y3) }
                   SwapVar:=X2;
                   X2:=X3;
                   X3:=SwapVar;
                   SwapVar:=Y2;
                   Y2:=Y3;
                   Y3:=SwapVar;
                 End;
  Incline:=(X2-X1) sal 6;
  If (Y2-Y1)<>0 then Incline:=Incline div (Y2-Y1)
           else Incline:=0;
  FixX:=X1 sal 6;
  For CntY:=Y1 to Y2 do
    Begin
      If (CntY>=ClipY1) and
        (CntY<=ClipY2) then Edge1[CntY]:=FixX sar
6;
      FixX:=FixX+Incline;
    End;
  Incline:=(X3-X1) sal 6;
  If (Y3-Y1)<>0 then Incline:=Incline div (Y3-Y1)
           else Incline:=0;
  FixX:=X1 sal 6;
  For CntY:=Y1 to Y3 do
    Begin
      If (CntY>=ClipY1) and
        (CntY<=ClipY2) then Edge2[CntY]:=FixX sar
6;
      FixX:=FixX+Incline;
    End;
  Incline:=(X2-X3) sal 6;
  If (Y2-Y3)<>0 then Incline:=Incline div (Y2-Y3)
           else Incline:=0;
  FixX:=X3 sal 6;
  For CntY:=Y3 to Y2 do
    Begin
      If (CntY>=ClipY1) and
        (CntY<=ClipY2) then Edge2[CntY]:=FixX sar
6;
      FixX:=FixX+Incline;
    End;
{ Cliiiiiiiiping Y }
  If Y1<ClipY1 then Y1:=ClipY1;
  If Y2>ClipY2 then Y2:=ClipY2;
  For CntY:=Y1 to Y2 do
    Begin
      XH1:=Edge1[CntY];
      XH2:=Edge2[CntY];
      If XH1>XH2 then Begin { Changes (XH1,YH1)
(XH2,YH2) }
                   SwapVar:=XH1;
                   XH1:=XH2;
                   XH2:=SwapVar;
                 End;
      Asm
        mov  bx,XH1
        mov  cx,XH2
        cmp  bx,ClipX1      { First, Clip X  }
        jg   @@NoClip1
        mov  bx,ClipX1
        cmp  bx,ClipX2
        jl   @@NoClip2
        mov  bx,ClipX2
      @@NoClip2:
        cmp  cx,ClipX1
        jg   @@NoClip3
        mov  cx,ClipX1
      @@NoClip3:
        cmp  cx,ClipX2
        jl   @@NoClip4
        mov  cx,ClipX2
      @@NoClip4:
        mov  ax,SegDes
        mov  es,ax
        mov  al,Col
        mov  ah,al
        push ax
        db   66h;shl  ax,16
        pop  ax
        mov  di,CntY
        mov  dx,di
        xchg dh,dl
        shl  di,6
        add  di,dx
        add  di,bx
        sub  cx,bx        { Lon }
        jz   @@NoLine
        mov  dx,cx
        shr  cx,2
        db   66h;rep  stosw
        and  dx,3
        jz   @@NoLine
        mov  cx,dx
        rep  stosb
@@NoLine:
      End;
    End;
End;
```

PROGRAMACIÓN DE LA ROM-BIOS (II)

Enrique de Alarcón

En el número anterior se explicaron muchos conceptos importantes y datos técnicos de interés relacionados con la ROM-BIOS, necesarios para todo programador de bajo nivel. En el presente artículo se explican a fondo todos sus servicios.

En el número anterior de la revista apareció la primera parte del presente artículo, y en el mismo se detallaron todos los conceptos básicos relacionados con la ROM-BIOS, explicando los diferentes tipos que hay, las áreas de datos que posee (con un detallado mapa de la misma), detalles sobre la CMOS, la SHADOW RAM, y un esquema de todos los servicios que pone la BIOS a disposición del programador. En esta nueva entrega se realizará una completa explicación de dichos servicios por grupos de utilidad, con lo cual el programador dispondrá de la información necesaria para poder acceder a todos ellos y usarlos a voluntad dentro de sus propias aplicaciones.

DATOS GLOBALES

A todos los servicios BIOS se accede mediante el uso de interrupciones software, o sea, mediante una instrucción ensamblador tipo *INT xx*, donde <xx> es el número de interrupción. El número de servicio que se quiera ejecutar de la misma se deberá indicar siempre en el registro AH. Así mismo, los parámetros de entrada que requiera la función se almacenan también en registros de la CPU (AL, BX, CX...), de forma que no se hace necesario el uso de la PILA de acumuladores (si no conoce el concepto de PILA, no necesita conocerlo para el tema del artículo). Por otra parte, los posibles errores que puedan devolver las funciones, si es que devuelven alguno, siempre se pueden detectar porque la bandera CF (*carry flag*) de la CPU está a 1 al volver de la ejecución del servicio (si CF=0, no hay error), y en AH se contiene el código de error.

Los tres tipos de BIOS (XT, AT y PS/2) no soportan todas las funciones detalladas, por eso, en casos especiales de funciones especificas de sólo una o dos clases de BIOS se informa por cuáles están soportadas. En caso de no indicar nada es que es soportada por todas.

INT 05h, HARDCOPY

Esta interrupción no posee más que una función, y sirve para imprimir el contenido de la pantalla (que debe estar en modo texto) de forma que se realiza una copia de pantalla sobre papel. Esta función se llama comúnmente <*Volcado de pantalla*> y hace años era muy usada, aunque actualmente está bastante desfasada en cuanto a utilidad (¿ quién va a imprimir, por ejemplo, un documento a "pantallazos"?). Esta interrupción puede ser llamada directamente desde el DOS pulsando la tecla <*IMPR PANT*>. Una vez llamada la función, se puede saber el resultado de la operación leyendo el byte de la dirección 0050h:0000h, donde el valor 0 indica sin error, 1 indica que esta imprimiendo y Ffh indica que se ha producido un error al imprimir.

INT 10h, ACCESO A PANTALLA

La interrupción 10h da control y acceso a la tarjeta de vídeo que esté instalada. Hay 23 servicios disponibles, y se detallan a continuación:

- *AH=00h*. Cambiar modo de pantalla : Permite cambiar el modo de funcionamiento de la pantalla entre uno de los estándar (CGA/EGA/MCGA/VGA) soportados por la BIOS, que incluye tanto modos de texto como gráficos. Entrada: AL=Nuevo modo. Salida: No hay salida. Algunos ejemplos de modos son: AL=0, modo texto de 40 por 25; AL=2, modo texto 80 por 25 color; AL=13h, modo gráfico 320x200 con 256 colores.
- *AH=01h*. Cambiar líneas del cursor:

Permite modificar las líneas que forman el cursor. Entrada: CH=Línea inicial (0-16), CL=Línea final (0-16). Salida: no devuelve nada.

- *AH=02h.* Situar cursor: Permite situar el puntero del cursor en las coordenadas X,Y que se le indiquen (las coordenadas son las unidades de carácter). Pudiendo así escribir textos con funciones BIOS donde se desee. Entrada: DH=Fila (0-24), DL=Columna (0-79), BH= Página donde está. Salida: No hay.
- *AH=03h.* Lee posición del cursor: Devuelve la posición del cursor de la página activa. Entrada: BH=Número de la página. Salida: DH=Fila, DL=Columna, CH=Línea incial cursor, CL=Línea final cursor.
- *AH=04h.* Sitúa lápiz óptico: Da la posición actual del lápiz óptico de pantalla (si es que hay). Entrada: No hay. Salida: AH=0 si no presente, AH=1 si lápiz pulsado, BX=Coordenada X en modo gráfico, CH= Coordenada Y en modo gráfico (modos 04h-06h), CL=Coordenada Y en modo gráfico (modos 0Dh-10h), DH=Coordenada Y en modo texto, DL=Coordenada X en modo texto.
- *AH=05h.* Cambia la página activa: Cambia la página de texto visible en el monitor y sobre la cual operan las funciones BIOS. Entrada: AL=Nueva página. Salida: No hay.
- *AH=06h.* Scroll arriba de pantalla: Desplaza todo el contenido de la ventana indicada de la pantalla X líneas hacia arriba. Las zonas de pantalla que se dejan vacías (última línea) se rellenan con el carácter indicado. Entrada: AL=Número de líneas a desplazar, CH=Fila inicial ventana a desplazar (Y0), CL=Columna inicial (X0), DH=Última línea (Y1), DL=Última Columna (X1), BH=Carácter de relleno en líneas sin contenido. Salida: No hay.
- *AH=07h.* Scroll abajo de pantalla: Desplaza todo el contenido de la ventana indicada una línea hacia abajo. Las zonas de pantalla que se dejan vacías (primeras líneas) se rellenan con el carácter indicado. Entrada: AL=Número de líneas a desplazar, CH=Fila inicial ventana a desplazar (Y0), CL=Columna inicial (X0), DH=Última línea (Y1), DL= Última Columna (X1), BH=Carácter de relleno en líneas sin contenido. Salida: No hay.
- *AH=08h.* Leer carácter/atributo de pantalla: Lee el carácter que hay escrito en la posición actual del cursor y su atributo. Entrada: BH=Página de la que leer. Salida: AL=Carácter que hay en la posición del cursor, AH= Atributo que contiene.
- *AH=09h.* Escribe carácteres/atributos: Escribe en la posición actual del cursor de la página que se desee el carácter y atributo indicados. Entrada: BH=Página destino, BL=Atributo del carácter (color en modo color), CX=Número de carácteres a escribir, AL=Carácter a escribir. Salida: No hay.
- *AH=0Ah.* Escribe carácter: Sólo escribe un carácter en la posición actual del cursor de la página indicada. Entrada: BH=Página, AL= Carácter. Salida: No hay.
- *AH=0Bh.* Modifica paleta de color: Reasigna el valor de un color de la paleta. Entrada: BH=Número de color, BL=Nuevo valor de color. Salida: No hay. Este sistema funciona sólo en los modos CGA de 320x200 de 4 colores.
- *AH=0Ch.* Escribe pixel: Escribe un pixel (un punto) del color deseado en la coordenada indicada de la pantalla en modo gráfico. Entrada: DX=Fila (Y), CX=Columna(X), AL=Color. Si color es mayor de 127, se realiza una operación XOR en el pixel indicado. Salida: No hay
- *AH=0Dh.* Leer pixel: Lee el color del pixel indicado de la pantalla (sólo posible en modo gráfico). Entrada: DX=Fila (Y), CX=Columna (X). Salida: AL=Color del punto de pantalla indicado.
- *AH=0Eh.* Escribe carácter y avance de cursor: Realiza la misma operación que la función 10h pero, además, el cursor avanza una posición (pasa a la posición siguiente). Entrada: AL=Carácter a escribir, BL=Atributo del carácter (color en modo gráfico), BH=Página sobre la que operar. Salida: No hay.
- *AH=0Fh.* Leer estado pantalla: Devuelve información de la pantalla, que incluye el modo de vídeo activo, las columnas que posee dicho modo y la página activa. Entrada: No hay. Salida: AL=Modo de vídeo, AH=Columnas, BH=Página activa.
- *AH=10h.* Color: Esta función posee numerosas subfunciones destinadas a controlar la paleta de las tarjetas tipo VGA (subfunciones 00h hasta 1Bh), y con las cuales se puede desde cambiar el borde de la pantalla hasta modificar los registros de la VGA. Debido a su extensión y a que forma parte de la programación de la VGA a bajo nivel, no se detallará más.
- *AH=11h.* Fuentes: Esta función contiene subfunciones de la 00h hasta la 30h, y están destinadas a controlar las fuentes ROM de la tarjeta de vídeo que se usan para escribir por la BIOS. Funcionan internamente a base de reprogramar el controlador de vídeo a nivel de registros.
- *AH=12h.* Configuración de vídeo: Posee nueve subfunciones destinadas a controlar diversos aspectos generales de la tarjeta de vídeo. Se puede, por ejemplo, desactivar la imagen de pantalla, cambiar el ratio de refresco del monitor, etc...
- *AH=13h.* Escribir cadena en pantalla: Envía la cadena especificada a la memoria de vídeo. Entrada: AL=Modo de escritura. Si AL=0, la cadena sólo contiene los carácteres (en BL el atributo) y el cursor no se desplaza; si AL=1, la cadena a escribir sólo contiene carácteres (atributo global en BL) y se desplaza el cursor tras escribir; si AL=2, la cadena contiene caracteres con sus respectivos atributos adjuntos (o sea, que los bytes pares son carácter y los impares son atributo) y no se desplaza el cursor; si AL=3, igual que AL=2 pero si desplaza el cursor. Salida: No hay.
- *AH=1Ah.* Información adaptador: Son dos subfunciones (AL=0/1), y devuelven información sobre el adaptador de vídeo. Entrada: AL=Subfunción, BH=Código de display inactivo, BL=Código de display activo (BH y BL sólo si AL=1). Salida: AL=1Ah si subfunción soportada, y si AL era 0h; BH/BL=Código inactivo/activo display.
- *AH=1Bh.* Información modo vídeo: Da un buffer de 64 bytes con información acerca del modo de vídeo activo. Entrada: BX=00h (siempre), ES:DI=Dirección del buffer de 64 bytes de memoria donde almacenar información. Salida: Información obtenida en ES:DI.
- *AH=1Ch.* Salvar/restaurar registros: Sirve para salvar y restaurar los registros de la tarjeta de vídeo del DAC

en/de memoria. Posee dos subfunciones (AL=0-3, leer estado buffer, salvar estado, restaurar estado).

INT 13h, ACCESO A DISCO

La interrupción BIOS 13h proporciona al programador todas las funciones básicas de acceso a las unidades de disco. En total hay veintisiete servicios, y son los siguientes:

- *AH=00h*. Resetear controladora del disco: Inicializa la controladora del disco duro y debe ser llamada después de cualquier operación de lectura/escritura que haya devuelto un error. Entrada: No hay parámetros. Salida: No hay.
- *AH=01h*. Último error: Devuelve el resultado de error que produjo la última función BIOS de disco que se haya ejecutado antes de ésta. Recordar que el valor que devuelve está tomado del área de comunicaciones de la BIOS en la dirección real 0040h:0074h (1 byte). Entrada: No hay parámetros. Salida: AL=Error del servicio anterior.
- *AH=02h*. Leer sectores: Permite leer X sectores del disco y almacenar su contenido en el buffer que se le indique. Entrada: DL= Unidad de disco, DH=Cara del disco, CH=pista, CL=Sector inicial, AL= número de sectores a leer, ES:BX=Dirección donde almacenar la información. Salida: AL=Sectores leídos.
- *AH=03h*. Escribir sectores: Permite escribir X sectores en el disco con el contenido del buffer de memoria que se le indique. Entrada: DL=Unidad de disco, DH=Cara del disco, CH=pista, CL=Sector inicial, AL=número de sectores a escribir, ES:BX=Dirección donde está almacenada la información a escribir. Salida: AL=Sectores escritos.
- *AH=04h*. Verificar sectores: Permite verificar el estado de X sectores del disco. Entrada: DL=Unidad de disco, DH=Cara del disco, CH=pista, CL=Sector inicial, AL=número de sectores a verificar. Salida: AL=Sectores verificados.
- *AH=05h*. Formatear pista: Permite formatear una pista del disco. Entrada: AL=Sectores, DL=Unidad de disco, DH=Número de la cara, CH=Número de pista que se quiere formatear. Salida: No hay.
- *AH=06h*. Formatear pista con indicadores: Permite formatear una pista y registrar los sectores defectuosos. Parámetros iguales a los de la función anterior.
- *AH=07h*. Formatear disco entero: Permite realizar un formateo de la unidad, ya sea de disquete o disco duro.
- AH=08h. Información unidad: Entrada: DL=Unidad de disco de la que obtener información. Salida: BL=Tipo de disco, CH=8 Bits inferiores del número de cilindros, CL=Bits superiores del número de cilindros (bits 0-5 bits mas altos, 6-7 inferiores), DH=Número cabezales, DL=Número de drivers, ES:DI=Puntero a la tabla de parámetros del disco.
- *AH=09h*. Inicializa disco duro: Inicializa el disco duro según su tabla de datos. Entrada: DL=Unidad de disco. Salida: No hay
- *AH=0Ah*. Leer sectores de HD: Permite leer muchos sectores del disco duro en memoria usando un sistema de 4 bytes ECC por sector. Parámetros igual que en función 02h.
- *AH=0Bh*. Escribir sectores en HD: Permite escribir gran número de sectores en el disco duro con el contenido del buffer de memoria que se le indique. Parámetros igual que en función 03h. Los datos a escribir deben estar en formato 4 bytes ECC por sector. Los 2 bits superiores del número de 10 bits con la pista están en los 2 bits superiores de CL.
- *AH=0Ch*. Sitúa Cabezal de lectura/Escritura en HD: Entrada: CH=8 bits inferiores de la pista, CL=2 bits superiores de la pista en bits 6-7, DH=cabezal, DL=unidad de Disco duro. Salida: No hay.
- *AH=0Dh*. Inicializa hardware del HD: Resetea su controladora y vuelve a situar los cabezales al inicio. Entrada: DL=Unidad de HD. Salida:No hay
- *AH=0Eh*. Lee sector interno del HD: Lee el sector del *buffer* temporal interno del HD en memoria. Sólo soportado por BIOS XT. Entrada: DL=Unidad de HD, ES:BX=Puntero de destino donde almacenarlo. Salida: No hay.
- *AH=0Fh*. Escribe en sector interno del HD: Escribe en el sector del *buffer* temporal interno del HD el *buffer* de memoria indicado. Sólo soportado por BIOS XT. Entrada: DL=Unidad de HD, ES:BX=Puntero origen donde está almacenado el sector a escribir. Salida:No hay.
- *AH=10h*. Estado HD: Devuelve si está o no operacional el Disco duro. Entrada: DL=Disco duro a examinar. Salida: CF=1, disco inoperativo; AH=error.
- *AH=11h*. Recalibra HD: Entrada: DL=Unidad de HD a recalibrar.
- *AH=12h*. Diagnóstico RAM: Realiza un test de estado del sector interno del disco duro. Sólo en BIOS XT. Entrada/Salida: No hay.
- *AH=13h*. Diagnóstico Hardware: Realiza un test de funcionamiento del hardware del HD. Sólo en BIOS XT. Entrada/Salida: No hay.
- *AH=14h*. Diagnóstico Controladora: Realiza un test de funcionamiento de la controladora del HD. BIOS AT y PS/2. Entrada/Salida: No hay.
- *AH=15h*. Tipo de disco: Informa sobre qué tipo de disco hay en la unidad indicada. Sólo en BIOS AT y PS/2. Entrada: DL=Unidad de disco. Salida: AH=Estado disco. Si AH=0, Unidad no presente; Si AH=1, es disquetera sin posibilidad de cambio de disco; si AH=2, disquetera con capacidad de cambio de disco; si AH=3, es un disco duro. En este último caso, además, devuelve CX:DX=Número de sectores de 512 bytes.
- *AH=16h*. Cambio de disco: Detecta si se ha cambiado el disquete de la unidad de disco. Sólo en AT y PS/2. Entrada: DL=Unidad de disquete a examinar. Salida: CF=0/AH=00h si disco no cambiado, CF=1/AH=6h si se ha cambiado. Como se ve, si se ha cambiado el disco devuelve un código de error.
- *AH=17h*. Cambiar modo disquetera: Cambia el modo de funcionamiento de la disquetera indicada para que pueda usar discos del tipo especificado. Sólo en AT y PS/2. Entrada: DL=Unidad de disco, AL=Tipo de *disketera* a emular. Si AL=00h no es usado, si AL=01h usar discos de 320/360K en unidad de 360K, si AL=02h discos de 320/360 en unidad de 1.2Mb, si AL=03h usar disquetes de 1.2Mb en unidad de 1.2Mb, si AL=04h usar disquete de 720K en unidad de 720Kb. Salida: No hay.
- *AH=18h*. Cambiar características unidad: Cambia las características físicas de los disquetes a usar en la unidad

indicada. Sólo en AT y PS/2. Entrada: CH=Número de pistas, CL=Sectores por pista, DL=Unidad a modificar. Salida: ES:DI=Puntero a la tabla de parámetros de la unidad modificada.

- *AH=19h.* Aparcar cabezales: Desplaza los cabezales del disco duro a la zona central para evitar daños en los mismos. Se utiliza normalmente antes de cambiar el ordenador de lugar físico para evitar averías del disco duro, ya que los cabezales son muy sensibles a los golpes. Sólo en BIOS PS/2. Entrada: DL=Unidad de HD a aparcar. Salida: No hay.
- *Ah=1Ah.* Formatea HD tipo ESDI: Formatea todo un disco duro tipo ESDI, usados sólo en los PS/2. Entrada: AL=Puntero a bloque relativo, CL=Formato de bits modificadores. Salida: No hay.

Es importante saber que todas las funciones no sirven para todos los tipos de BIOS. Además, hay que tener presentes algunos datos para poder usar estas funciones. Lo primero y más importante es que hay unidades de disco que tardan bastante en arrancar el motor de giro y, por la tanto, la primera e incluso la segunda vez que se ejecute, por ejemplo, una función de lectura de sectores, puede no estar aún preparada la unidad para leer y devolver en consecuencia un error de lectura. Es por eso que cualquier operación debe ejecutarse normalmente un mínimo de tres veces antes de devolver un resultado real de la operación efectuada. Lo siguiente que se debe saber es que cuando se devuelve un error, ya sea durante uno de los tres intentos mencionados de repetición de una operación o en cualquier otro caso de error real, se deberá reinicializar la controladora de disco con la función 00h antes de volver a llamar a la función de lectura/escritura.

Por último, se deben conocer los 9 posibles errores que pueden devolver dichas funciones en AH.

a) Si AH=00h, no existe error.
b) Si AH=02h, es un disco defectuoso.
c) Si AH=03h, el disco está protegido contra escritura (es un disquete).
d) Si AH=04h, sector no encontrado (seguramente el disco es defectuoso).
e) Si AH=08h, error del DMA.
f) Si AH=10h, error de CRC .
g) Si AH=20h, la controladora de disco está dañada.
h) Si AH=40h, error de búsqueda.
i) Si AH=80h, límite de tiempo sobrepasado (disco defectuoso o unidad no preparada). Este último error se da normalmente durante los 3 intentos de operación sobre los discos antes mencionados.

INT 14h, PUERTOS SERIE

Este servicio permite establecer comunicación de datos entre dos ordenadores de forma fácil y sencilla, ya sea unidos directamente por un cable o por medio de un módem a través de la línea telefónica. Los servicios que ofrece son muy limitados, y será necesario conocer cómo programar directamente las UART para realizar según qué operaciones, pero pueden ayudar en su programación global. A continuación se explican las seis funciones que ofrece:

- *AH=0h.* Inicializa puerto serie: Prepara el puerto indicado para fucionar en las condiciones que se le indiquen. Entrada: DX=Número de puerto serie; AL=Parámetros, donde los bits 0-1 son la longitud de cada carácter (5, 6, 7 u 8 bits); bit 2=Número de bits finales; bits 3-4=Paridad (no/impar/par); bits 5-7=Velocidad de comunicación (000=110 baudios, 001=150, 010=300, 011=600, 100=1200, 101=2400, 110=4800 baudios, 111=9600 baudios). Salida: AX=Estado del puerto.
- *AH=1h.* Envía carácter: Envía un carácter al módem, el cual lo transmitirá cuando la línea esté preparada para ello. Entrada: DX= Número de puerto serie al que enviar, AL=Carácter a enviar. Salida: AH=Estado de la línea.
- *AH=02h.* Recibe carácter: Recibe un carácter del módem. Entrada: DX=Número de puerto del que se quiere recibir. Salida: AL=Carácter recibido, AH=Estado de la línea.
- *AH=03h.* Lee estado puerto serie: Devuelve una palabra (2 bytes) con el estado del puerto serie. Entrada: DX=Puerto del que se quiere recibir información. Salida: AX=Palabra de estado, AH=Estado de la línea, AH=Estado del módem.
- *AH=04h.* Inicializa extendido: Funciona de forma similar a la función 00h, pero ofrece más posibilidades. Sólo en BIOS PS/2. Entrada: AL=Bandera de *break*, BH=Paridad, BL=Bits de parada (0=1 y 1=2 para chars de 6-7 bits/1.5 para *chars* de 5 bits), CH=Bits por carácter (5,6,7 u 8 bits/car), CL=Velocidad de transferencia (0=110, 1=150, 2=300, 3=600, 4=1200, 5=2400, 6=4800, 7=9600, 8=19200), DX=Puerto a usar. Salida: AX=Palabra de estado, AH=Estado de la línea, AH=Estado del módem.
- *AH=05h.* Escritura/lectura de módem: Permite escribir o leer del módem. Sólo en BIOS PS/2. Entrada: AL=Subfunción (0=Leer Registro de control del módem, 1=Escribir registro de control), BL=Registro de módem (si subfunción seleccionada es 1). Salida: BL=Registro de módem si es subfunción 0, no hay si es subfunción 1.

A continuación se detalla el contenido de la palabra que nos devuelve esta función:

- *AH,* Estado puerto serie:

a) Bit 0=Dato preparado.Bit
b) Bit 1=Sobrecarga.Bit
c) Bit 2=Error en paridad.Bit
d) Bit 3=Error del formato.Bit
e) Bit 4=*Break* por interrupción.Bit
f) Bit 5=THR Vacío.Bit
g) Bit 6=TSR Vacío.Bit
h) Bit 7=Tiempo excedido.

- *AL,* Estado del módem:

a) Bit 0=CTS, Preparado para enviar.Bit
b) Bit 1=DSR, Dato preparado para recibir.Bit
c) Bit 2=RI, Indicador de llamada.Bit
d) Bit 3=DCD, Línea presente.
e) Bit 4=CTS, variación en señal (-Bit 0).
f) Bit 5=DSR, variación en señal (-Bit 1).
g) Bit 6=RI, variación en señal (-Bit 2).
h) Bit 7=DCD, variación en señal (-Bit 3).

Los datos relacionados con módem/UART no se detallan por quedar fuera de tema y/o por que ya se han detallado en anteriores artículos.

INT 15h, MISCELÁNEA

La interrupción 15h contiene múltiples funciones destinadas a muy diversas tareas. Desde usar memoria XMS hasta leer el joystick. De todas ellas se han detallado las que pueden ser de utilidad:

• *AH=84h, DX=0.* Estado botones joysticks: Con esta función se obtiene el estado de los botones cero y uno de los dos joysticks que pueda haber instalados en el sistema. Entrada: No hay. Salida: AH-Bit 7=Botón 0/Joystick A activo, Bit 6=Botón 1/Joystick A activo, Bit 5=Botón 0/Joystick B activo, Bit 4=Botón 1/Joystick B activo.

• *AH=84h, DX=1.* Estado Palanca Joysticks: Devuelve la posición de la palanca de los dos joysticks. Entrada: No hay. Salida: AX=Posición X/joystick A, BX=Posición Y/joystick A, CX=Posición X/joystick B, DX=Posición Y/joystick B.

En los dos servicios, si se devuelve el *flag* CF=1, significa que no hay joystick alguno conectado. Igualmente se puede saber esto sin necesidad de usar estas funciones examinando si han sido detectados durante la inicialización del ordenador (examinar palabra información hardware de la BIOS, explicada en el anterior artículo).

INT 16h, TECLADO

Esta interrupción da acceso a las funciones básicas de control de teclado. La ubicación de los *búffers* sobre los que se basan para funcionar ya se detalló en el anterior artículo de la BIOS. Sólo hay cinco servicios y son los siguientes:

• *AH=00h.* Leer tecla del *buffer* y avanzar puntero: Lee la primera tecla pendiente almacenada en el *buffer* del teclado de la BIOS (con capacidad para 16 pulsaciones) y sitúa el puntero en el siguiente carácter. La tecla que devuelve posee 2 bytes, y puede contener una combinación de teclas especiales o un carácter normal. En caso de no haber tecla alguna en espera dentro del *buffer*, la función espera hasta que se pulse una tecla. Entrada: No hay . Salida: AH=Id de la tecla, AL=Código ASCII de la tecla.

• *AH=01h.* Estado del *buffer*: Devuelve si está o no vacío el *buffer* del teclado de la BIOS.

Entrada: No hay. Salida: Si la bandera de la CPU ZF=1, está vacío; si está a cero, contiene pulsaciones; entonces AH=Id de pulsación, AL=Código de carácter.

• *AH=02h.* Estado teclado: Devuelve un byte con información sobre el estado de teclas especiales del teclado. Dicho byte se extrae del área RAM de comunicaciones de la BIOS y se detalló su contenido en el artículo que apareció en el número anterior de esta revista. Entrada: No hay. Salida: AL=Byte con información.

• *AH=03h.* Ratio de repetición: Permite, al igual que se hace desde el menú de la BIOS, cambiar los carácteres por segundo en modo repetición de pulsación y el tiempo entre pulsación de la primera tecla y el inicio de su repetición. Entrada: AL=05h, BH=Tiempo antes de repetición, BL=Velocidad de repetición. Salida: No hay.

• *AH=05h.* Insertar pulsación en buffer: Inserta el carácter y el atributo indicado en el *buffer* de teclado como si de una pulsación real se tratara. Sólo BIOS AT y PS/2. Entrada: CH=código extendido carácter, CL=Carácter. Salida: No hay.

INT 17h, LA IMPRESORA

Para controlar la impresora, solamente se dispone de 3 servicios básicos, pero que son suficientes para usar la misma. A continuación se detalla su uso:

• *AH=00h.* Imprimir carácter: Envía un carácter a la impresora (o a su *buffer* si lo posee) e imprime el carácter (bajo las condiciones de impresión actuales programadas). Entrada: AL=Carácter a imprimir. Salida: AH=Bytes de estado de la impresora. Más adelante se detallará su contenido.

• *AH=01h.* Activar impresora: Pone como activa la impresora que se le indique (de las cuatro máximas posibles conectadas al sistema) y devuelve información sobre la misma. Entrada: DX=Impresora a activar. Salida: AH=Información sobre la impresora activada.

• *AH=02h.* Leer estado impresora: Se utiliza sólo para obtener el byte de estado de la impresora que se le indique. Entrada: DX=Impresora de la que obtener información. Salida: AH=Bytes de estado de la impresora indicada.

A continuación se detalla qué información da el byte de estado de la impresora indicado en las funciones anteriores:

a) Bit 0=Tiempo excedido (1=error).
b) Bit 1-2=No usados.
c) Bit 3=Error de Entrada/salida (0=error).
d) Bit 4=Impresora en línea (1=si).
e) Bit 5=Estado alimentación papel (1=sin papel).
f) Bit 6=Existencia impresora (1=presente).
g) Bit 7=Estado funcionamiento (1=No ocupada, 0=imprimiendo).

INT 1Ah, RELOJ

Son diez funciones destinadas a manejar todos los aspectos del reloj interno del sistema. Todas estas funciones sólo están en los modelos AT y PS/2. Son las siguientes:

• *AH=00h.* Leer contador: Devuelve el valor actual del contador interno del sistema y si ha pasado el medio día o la media noche desde su última lectura. Entrada: AL=Si cambio de AM a PM presente, CX:DX=Contenido del contador. Salida: No hay.

• *AH=01h.* Escribir contador: Modifica el contenido del contador interno del sistema. Entrada: CX:DX=Número 32 bits con nuevo valor para el contador. Salida: No hay.

• *AH=02h.* Leer hora del reloj: Sólo en AT y PS/2. Entrada: No hay. Salida: CH=Hora en BCD, CL=Minutos en BCD, DH=Segundos en BCD, DL=tipo de hora, CF=0 si el reloj está funcionando.

• *AH=03h.* Cambiar hora: Entrada: CH=Hora en BCD, CL=Minutos en BCD, DH=Segundos, DL=Tipo de hora. Salida: No hay.

• *AH=04h.* Leer fecha: Entrada: No hay. Salida: CH=Siglo DC, CL=Año en BCD, DH=Mes en BCD, DL=Día en BCD, CF=0 si reloj funcionando.

• *AH=05h.* Leer fecha: Entrada: CH=Siglo DC, CL=Año en BCD, DH=Mes en BCD, DL=Día en BCD. Salida: no hay.

• *AH=06h.* Poner alarma: Entrada: CH=Hora alarma, CL=Minutos alarma, DH=Segundos. Salida: No hay

• *AH=07h.* Resetear alarma: Entrada/Salida: No hay.

• *AH=0Ah.* Días pasados: Devuelve los días pasados desde el 1-1-1980. Entrada: No hay. Salida: CX=Días pasados.

• *AH=0Bh.* Cambiar contador días: Escribe nuevo número de días pasados desde el 1-1-1980. Entrada: CX=Nuevo número. Salida: No hay.

¡¡NO ESPERES MAS!!

PARA ACCEDER A LAS AUTOPISTAS DE LA INFORMACION

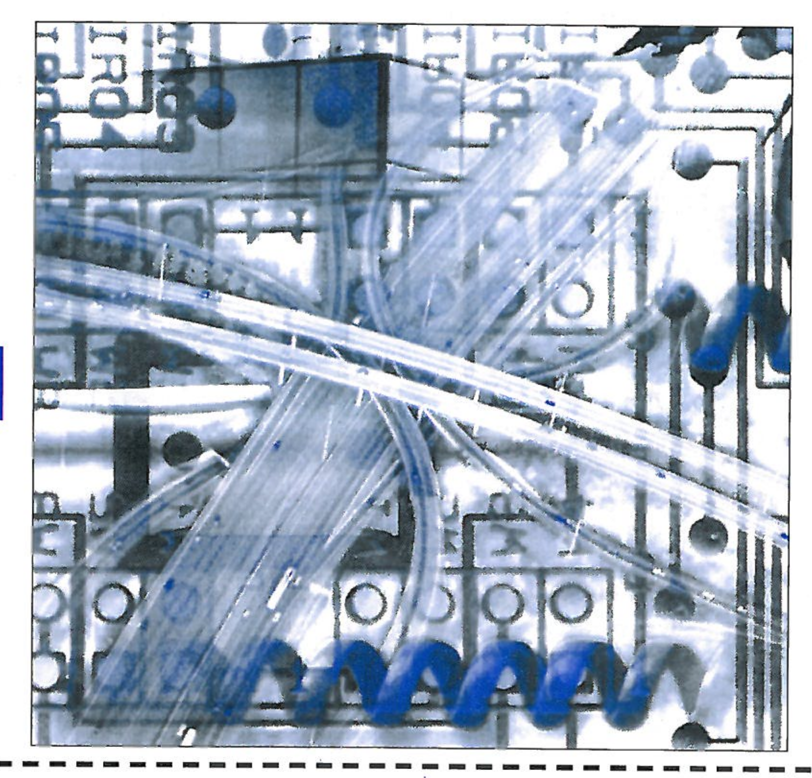
¡CONECTATE!

PORQUE LA INFORMACION ES PODER Y PORQUE QUIERES SABER MAS, NO PUEDES ESTAR DESCONECTADO DE MAS DE 40.000.000 DE PERSONAS DE TODO EL MUNDO QUE PIENSAN COMO TU. Y QUE SABEN, QUE LA INFORMATICA NOS AYUDA A ELIMINAR BARRERAS , Y A ENTENDERNOS MAS Y MEJOR.

AHORA PUEDES CONECTARTE A INTERNET A UN PRECIO EXCEPCIONAL

¡¡¡ 1.995 PTAS. AL MES !!!

- SIN CUOTA DE CONEXION.
- SIN LIMITE DE TIEMPO.
- SIN RETRASOS NI ATASCOS.
- INCLUYENDO SOFTWARE INFOVIA-INTERNET



Envíanos este cupón de solicitud de conexión y recibirás el software y una domiciliación bancaria. Remitir a Eurotex c/ Augusto Figueroa 32-4º E Madrid-28004 - Tfno. (91) 523 18 93 (de 19 a 21h.)

Nombre. . . . .

Dirección . . . .

Población . . . . Provincia . . . .

CP. . . . . Teléfono . . . .

RESOLUCIONES VIRTUALES

Santiago Romero, Miguel Cubas y Vicente Cubas

Una vez finalizado el tema de los registros, comienza ahora el tema de las resoluciones de pantallas, que permitirán hacer scrolles multidireccionales y organizar la memoria de la VGA a nuestro gusto

La tarjeta de vídeo dispone de al menos 256K de memoria, como se pudo ver en anteriores artículos, con lo que hacen un total de 4 pantallas en 320x200 (y un poco más, ya que un segmento son 65535 bytes, y 320x200 son 64.000).

Lo que se viene a decir en este apartado es que esta memoria se puede manipular con el fin de obtener unas dimensiones determinadas de memoria de video para el MODO Xy 13X. Es decir, que si se tienen 4 pantallas se pueden organizar de varias maneras. Un ejemplo sería programar el MODO 13X para unas dimensiones virtuales de 640x400 con lo que se obtendrían 2x2 pantallas. Puede verse en la figura 1 cómo están organizadas las pantallas verticalmente. Esta es la posición inicial que presenta el Modo 13X al ser iniciado, y que luego puede ser modificado a gusto del programador de acuerdo con el programa que vaya a realizar. También se indican en las figuras 2 y 3 los restantes modos principales de los que se puede disponer en el Modo 13X.

Hay que tener en cuenta que la forma de calcular el *offset* para dibujar un punto en pantalla no es la misma en todas las dimensiones, con lo que se tendrá que programar una rutina diferente para cada modalidad de tamaño, debido a que en un modo de 1x4 pantallas no hay el mismo número de bytes horizontales (80) que en uno de 2x2 (160), como se puede ver en las figuras.

Esto de programar una rutina para cada modo de pantalla es una difícil tarea debido a la gran variedad de tamaños que existen, por lo tanto no queda más remedio que crear para cada juego, demo o utilidad gráfica una rutina que funcione en éste, o bién hacer una rutina que se le pueda pasar un parámetro indicando con qué modalidad se está trabajando. Más adelante se proporcionará una función que calculará los *offsets* de diferente forma para cada dimensión de pantalla.

Nótese que la manera de programar la organización de las páginas no afecta a la resolución de pantalla, sino a la manera en que se organizan las páginas virtuales dentro de la memoria, organización que puede ser beneficiosa para determinado tipo de efectos.

VAMOS A LA PRÁCTICA

La manera de programar los registros para obtener estas dimensiones es muy sencilla, tan sólo hay que acceder a un registro y mandar el valor correspondiente para acceder a las nuevas resoluciones virtuales. Con esto se accede de otra manera a la memoria, disponiendo de un nuevo ancho y alto.

El registro de la VGA que se utiliza para conseguir esto es el *Offset Register*, del *Cathode Ray Tube Controller*(CRTC), cuya definición se puede encontrar en las tablas de registros publicadas en el número anterior de Sólo Programadores, y que aparecen en la tabla 1.

Con la función RegisterOut() de la librería VGAREGS.H, su modificación se reduce a:

RegisterOut(CRTC_ADDR, 0x13, valor);

No obstante, puede hacerse también sin la librería, tal y como se explicó en el pasado artículo para reprogramar el CRTC:

Este número entero indica la anchura lógica (contando ya los planos) que tiene cada *Scan Line* (cada línea horizontal de la pantalla) dividido entre dos. De esta manera, el valor inicial que tiene este registro al inicializar el modo X es 40, ya que se dispone de 80 bytes horizontales

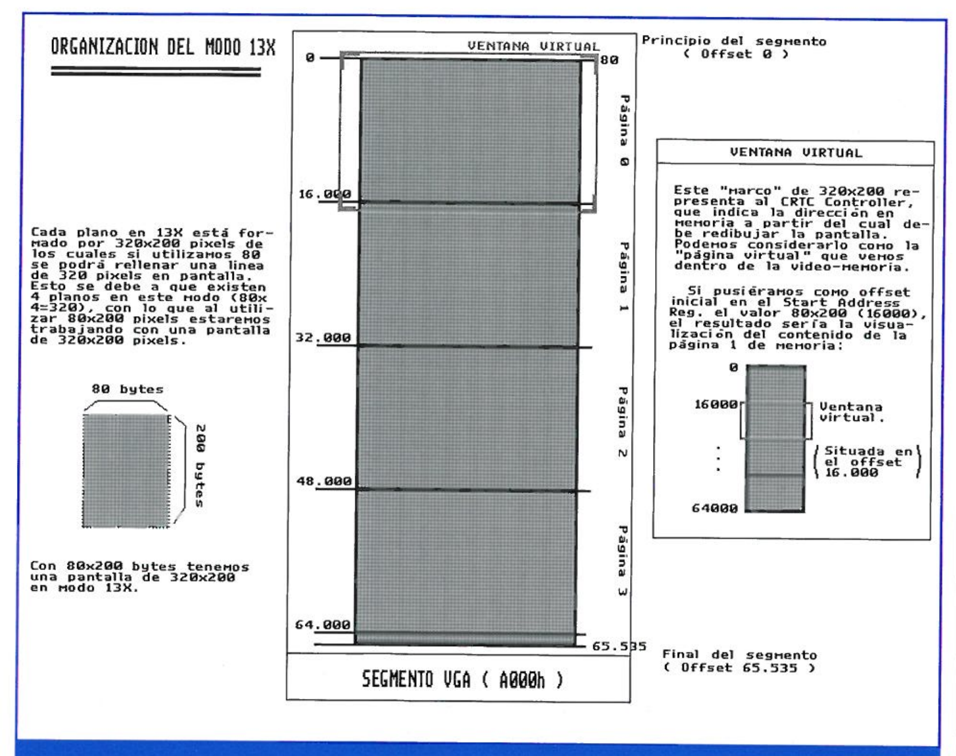


Figura 1: Organización del modo X (1x4)

LISTADO 1

```
/*
SetDimX( char DIM )

Selección del sistema de organización de memoria
en forma de pantallas virtuales. Se le pasa el valor
YA DIVIDIDO entre 2. El motivo de la división entre
dos está explicado en el artículo. También pueden
usarse los defines como parámetros:

 Valores de DIM:
    DIM1x4 —> variable 'anchura' = 40 (28h)
    DIM2x2 —> variable 'anchura' = 80 (50h)
    DIM4x1 —> variable 'anchura' = 160 (A0h)
*/

void SetDimX ( unsigned char anchura )
{

    if( anchura == DIM1x4 ) modoactual = 0;
    else if( anchura == DIM2x2 ) modoactual = 1;
    else modoactual = 2;

    asm mov dx , 0x3d4       /* CRTC_ADDR */
    asm mov al , 0x13        /* offset register */
     asm out dx , al         /* valor 13h al puerto
0x3d4 */

    asm inc dx               /* cambiamos el puerto
al 0x3d5 */
    asm mov al ,[anchura]
    asm out dx , al          /* valor anchura al puer-
to 0x3d5 */

}
```

Cambio de resoluciones virtuales.

por cada línea, como se puede ver en la figura 1. Si se deseara colocar el modo X en un modo de, por ejemplo, 2x2 pantallas, se tendrían, como se puede ver en la figura 2, 160 bytes por cada línea (aunque la pantalla siga teniendo 80 bytes por página). Como hay que partirlo por dos, resultaría lo siguiente:

RegisterOut(CRTC_ADDR, 0x13, (160/2));

Pero, ¿por qué hay que dividir entre dos? Muy sencillo, es debido a la simple limitación de que los puertos tan sólo pueden manejar bytes, así que si se quisiera enviar el valor 320 (4x1 pantallas; 80 bytes*4 pantallas = 320), nos encontraríamos con un número mayor de 255. El sistema que se decidió pasa por dividir por 2 la anchura lógica y enviarlo al puerto:

valor = (anchura / 2);

Ya que en este artículo se va a trabajar en una resolución de 320x200, se utilizará la memoria de vídeo disponible para dimensionar la pantalla en tres modos principales, que son: 1x4, 2x2 y 4x1 pantallas, aunque el lector se puede crear sus propias resoluciones de páginas probando distintos valores con el *Offset Register*. En el listado 1 se puede observar cómo han sido programados estos 3 modos principales.

CALCULAR EL OFFSET

Como se ha dicho antes, la forma de calcular el *offset* para poner un punto en pantalla no es la misma para cada organización de memoria.

Si se desea poner un pixel en 1x4 pantallas, se dispone (ver figura 1) de 80 bytes por línea. En este caso, para calcular el *offset* del pixel en la posición X,Y se usaba la fórmula:

*Offs = (y*80) + (x/4)*

Si se inicializa, por ejemplo, el modo de 2x2 pantallas, aunque sólo haya 80 píxels por pantalla, la página virtual que se ha organizado (ver figura 2) obliga a saltar 160 bytes por línea, con lo que la fórmula quedaría así:

*Offs = (y*160) + (x/4)*

Esto supone tener una rutina para cada modo de pantalla, pero hay un cálculo que permite utilizar una sola rutina para los diferentes offsets a cal-

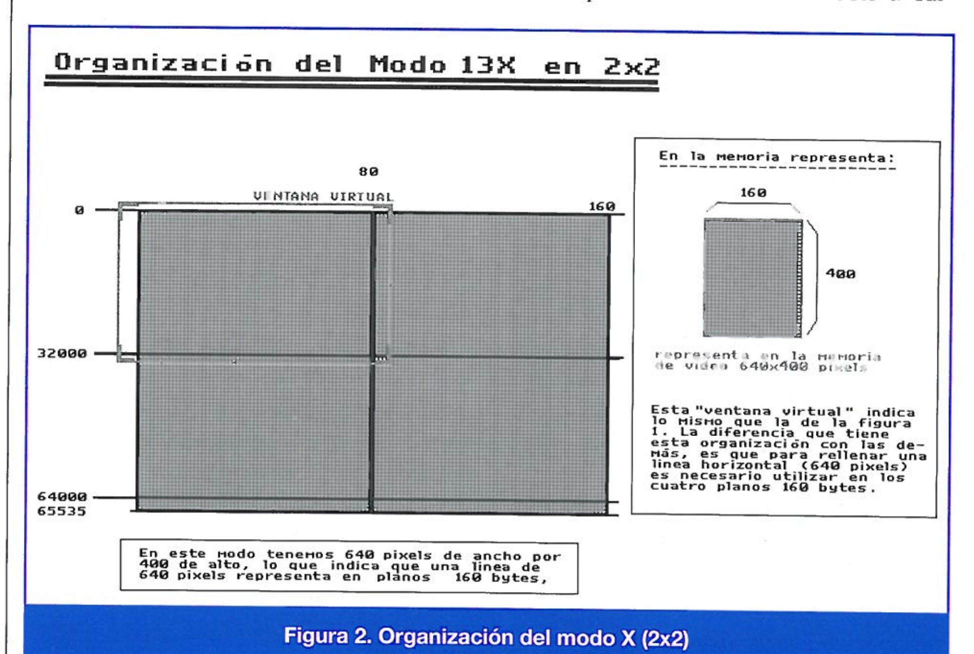


Figura 2. Organización del modo X (2x2)

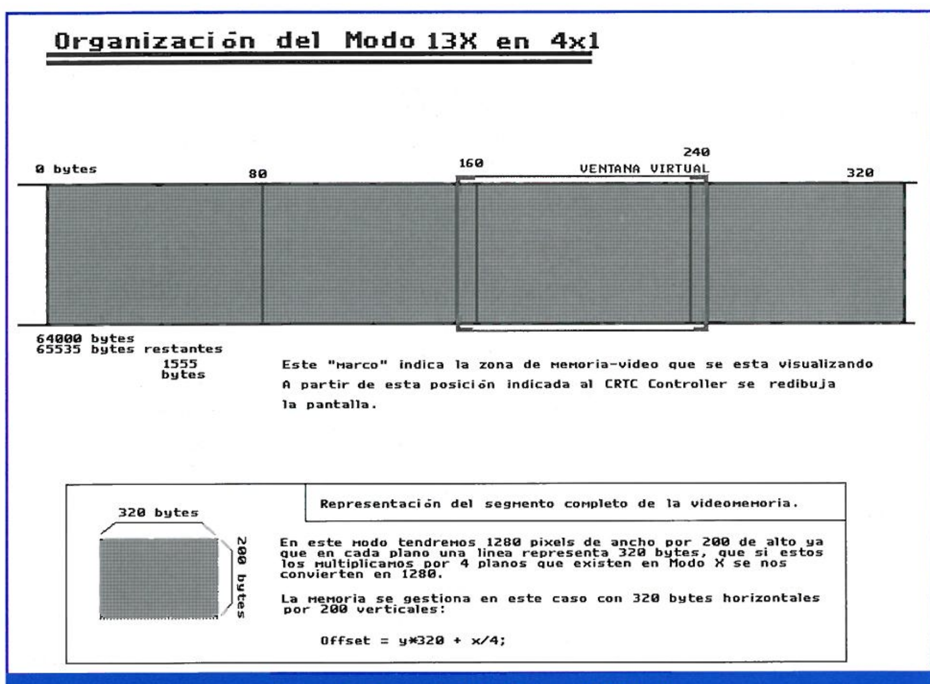


Figura 3. Organización del modo X (4x1)

LISTADO 2

```
void PutPixelX( int x, int y, char color )
{
unsigned int offs;

OutPortb( SEQU_ADDR , 0x02 ); /* función selec-
cionar plano */
OutPortb( SEQU_ADDR+1, 0x01 << ( x & 3 ) );

        /* calculamos el offset */
offs = (((y<<6) + (y<<4))<< Array_Desp[ modoac-
tual ] );
offs += (x>>2);

asm {
        push es
        mov ax , 0xa000
        mov es , ax
        mov di , offs
        mov al , color
        stosb
        pop es                  /* ponemos el
punto */
    }
}
```

Sistema de desplazamientos aplicado a PutPixel.

cular. Este cálculo permitirá utilizar los tres modos mencionados hasta ahora, que son los más comunes, de manera que si se desean utilizar otras dimensiones se tendrá que hacer una rutina para cada una de estas.

En el listado 2 se muestra la manera de calcular el *offset* en las diferentes organizaciones. Como se puede ver en este listado, estan definidos tres modos asignándoles a cada uno de ellos un número. Este número corresponderá a la variable *modoactual,* definida y utilizada en la rutina que calcula el offset en pantalla. A continuación hay definido un array de 4x3 que indica el *offset* de comienzo de cada página en estos modos. Aquí se pone el *offset* de cada página para los 3 modos de organización X, para agilizar los cálculos.

Antes de utilizar estas rutinas hay que llamar a las funciones SetModennn(); y SetDimX() indicándoles en qué modo se va a trabajar y la página que se utilizará para volcar los gráficos, para que actualicen los valores de las variables necesarias para nuestra nueva librería.

En este listado se ve el *array* que contiene tres valores que ayudarán a adaptar el *offset* calculado para la coordenada Y al modo que corresponda. Como puede verse, este *array* contiene los valores 0, 1 y 2. Lo que se hace es calcular *80*y + x*, y el resultado se multiplica por 1, 2 o 4 según el modo de pantalla que esté inicializado.

Es decir, si se tiene como coordenada Y el valor 30 y se calcula su offset para una página (80), habrá que multiplicar por cuatro para obtener el *offset* de Y en el modo 4x1.

El motivo por el que se seleccionan estos valores en funciones es cuestión de organización y tiempo de acceso, para no tener que pasar estos parámetros a las funciones y sufrir su introducción y extracción de la pila. Se definen de forma global y se inicializan una sola vez al comienzo del programa, ya que de lo contrario habría que asignar el valor cada vez que se llame a la función.

Una vez se hayan asignado los valores de *modoactual* y de *página,* se debe llamar a la función mandándoles las coordenadas X e Y y la función se encargará de devolver un valor que corresponderá al *offset* pedido. La manera de calcular el *offset* es muy sencilla, como se puede observar en el listado 2. Lo primero que se calcula es el *offset* para una pantalla de 320x200 sin contar con la coordenada X. Este valor tiene que multiplicarse por un número tal que se adapte al modo correspondiente. Es decir, si se tienen unas dimensiones de 4x1 habrá que multiplicar este *offset* por cuatro, ya que la anchura del área gráfica es cuatro veces más grande. Para evitar engorrosas multiplicaciones se usa la instrucción *SHL* (<<), de manera que, dependiendo del modo en el que se esté, se desplazará (y*80) tantas veces como sea necesario para multiplicar por 1, 2 o 4, dependiendo del valor de la variable *modoactual.*

De esta ingeniosa manera, por desplazamientos, se consigue calcular el *offset* para cualquier modo de pantalla.

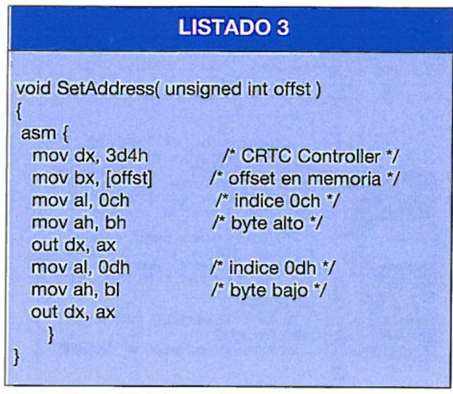
El siguiente paso es sumar a este resultado un número con tal de colocarse en el principio de la página seleccionada y en la coordenada Y que corresponda, calculada en el paso anterior. Finalmente al resultado de todo este proceso se le suma la coordenada X dividida entre cuatro, ya que se trabaja en MODO X, y hay que calcular y seleccionar el plano en el que cae cada pixel.

Para finalizar se devuelve el resultado al lugar donde ha sido llamada esta función, que puede ser utilizado para una función que ponga un pixel o para cualquier otra.

LAS PÁGINAS VIRTUALES

Veamos ahora de dónde salen estas páginas y de cuántas se dispone. En modo 13X, al gestionarse una pantalla completa con tan sólo 80 bytes, pues





LISTADO 3

```
void SetAddress( unsigned int offst )
{
 asm {
  mov dx, 3d4h          /* CRTC Controller */
  mov bx, [offst]       /* offset en memoria */
  mov al, 0ch           /* indice 0ch */
  mov ah, bh            /* byte alto */
  out dx, ax
  mov al, 0dh           /* indice 0dh */
  mov ah, bl            /* byte bajo */
  out dx, ax
    }
}
```

Función SetAddress( ).

estos 80 bytes tienen acceso a 4 pixels, se ve que la pantalla está formada por 80*200 = 16.000 bytes. Como el segmento de la vídeo memoria tiene 65.536 bytes, resulta que se dispone de 65.536 / 16.000 = 4'1 páginas virtuales, que pueden ser colocadas de cualquier manera (1x4, 2x2, 4x1 y, en definitiva, cualquier resolución que sume 4 pantallas, como 2.8x1.7).

En modo X, debido a las 40 líneas verticales extra, se dispone de 3'6 páginas, cuya mejor colocación podría ser 2x3.6, 2x1.9 y 4x0.9 páginas.

CRTC
UN EJEMPLO DE USO DEL CRTC:
SCROLL VERTICAL POR HARDWARE

Basándonos en las funciones que contiene el CRCT *Controller*, explicadas en el número anterior, pueden desarrollarse efectos profesionales que resultan más dificiles de realizar en el modo estándar 13h, tales como el *scroll* hardware vertical para modo X en sistema de 1x4 pantallas.

Intentar hacer un *scroll* mediante *buffers* virtuales y vuelques a pantalla de estos *buffers* puede llegar a requerir una compleja sincronización si se quiere evitar la diferencia de velocidad que existe entre los distintos modelos de PC, pero puede resultar sencillo si se ponen en práctica los conocimientos adquiridos sobre el *CRTC Controller* y se toma como base del *scroll* el retrazo, que prácticamente asegura la misma velocidad en distintos modelos, pues éste ocurre siempre entre 60 y 70 veces por segundo, según el monitor y modo de vídeo establecido.

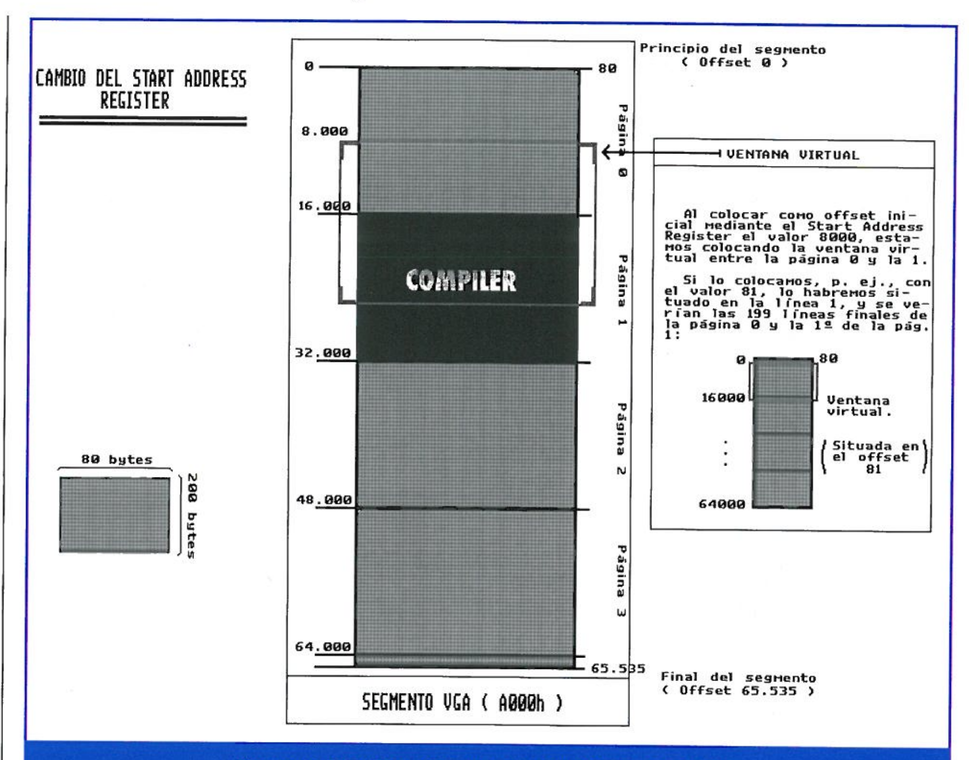


Figura 4. Modificación del Start Address Register.

CAMBIOS DE PÁGINA

Para comenzar, puede verse en la figura 1 la organización de la video-memoria (segmento A000h) cuando está inicializado el modo X en 320x200. Como puede verse, las cuatro páginas de que se dispone están colocadas verticalmente una sobre otra, de manera que los bytes que representan el gráfico de la página 0 están al principio del segmento, desde el *offset* 0 hasta el *offset* 16.000. A partir del *offset* 16.000 (80x200), comienzan los bytes que representan el gráfico de la página 1, y así sucesivamente se encuentran las siguientes páginas en el *offset* 32.000 (página 2) y 48.000 (página 3).

El haz de electrones, al redibujar la pantalla, comienza a leer bytes a partir del *offset* correspondiente a la página actual y los representa en pantalla con sus distintas tonalidades según los valores del DAC de video (según la paleta) que estén establecidos en ese momento. Si hubiese alguna manera de cambiar este *offset* por defecto a 80x200, por ejemplo, entonces lo que el haz de electrones redibujaría durante el siguiente ciclo, como se puede ver en la figura 1, sería la página 1, entendiendo como página 0 la que aparece por defecto para las operaciones gráficas.

La manera de hacer esto, de cambiar la dirección que se toma como la esquina superior izquierda de la "ventana virtual", dentro del marco de la vídeo memoria, la brinda el *CRTC Controller* mediante el registro *Start Address Register,* índices 0ch y 0dh. Al registro de índice 0ch (*Start Address High*) se le proporciona el byte alto del nuevo *offset,* y al que tiene como índice 0dh (*Start Address Low*) se le pasa el byte bajo. Hubo de ser dividido en dos registros porque es la mejor manera de pasar un valor de 16 bits (un offset) mediante puertos, que transmiten valores de 8 bits.

Gracias a las funciones que controla el CRTC se puede colocar la "ventana virtual" en cualquier lugar de la video

TABLA 1

Puerto: 3d4h, CRCT_ADDR
Index: 13h
Descripción:

3d4h index 13h (r/W): CRTC: Offset register

bit 0-7 Número de bytes por ScanLine (línea horizontal) partido por N, donde N es 2 para modos de sistemas de direccionamiento de bytes, 4 para modos de palabras y 8 para modos de dobles palabras. En Modo 13X y X hay que dividir por 2, pues es un modo de bytes

Offset Register, CRTC, Index 13h.

TABLA 2

Puerto: 3d4h, CRTC_ADDR
Index: 0Ch y 0Dh.
Descripción:

3d4h index Ch (r/W): CRTC: Start Address High Register:

bit 0-7 8 bits superiores de la dirección en memoria (offset lógico) que considerar como inicio al redibujar la pantalla.

3d4h index Dh (r/W): CRTC: Start Address Low Register

bit 0-7 8 bits inferiores de la dirección en memoria (offset lógico) que considerar como inicio al redibujar la pantalla.

Start Address Register, CRTC, Index 0Ch y 0Dh.

memoria, no sólo de página en página, sino que se puede seleccionar como *offset*, por ejemplo, el valor 8.000 (80x100) y visualizar en pantalla (figura 4) las 100 últimas líneas de la página 0 y las 100 primeras de la página 1. Esta última opción es la que se usará para realizar el *scroll* vertical.

LA FUNCION SETADDRESS

En la librería VGAREGS.H está disponible desde el número anterior la función *SetAddress()*, a la que se le pasa un valor de 16 bits que constituye el *offset* donde se desea colocar el "marco virtual".

Por supuesto, esta función está basada en el Start Address Register, como se puede ver en el listado 3.

La función *SetPage()* de MODOX.H utiliza el *Start Address Register* para colocar el marco virtual en cualquiera de las páginas, contando como página 0 la inicial, situadas en los *offsets* preparados en uno de los arrays globales de la librería.

EL ALGORITMO DE SCROLL

Imaginemos una imagen situada en la página 1 de pantalla en modo X. Esta imagen estaría situada, por lo tanto, a partir del *offset* 16.000 en la vídeo-RAM. Este esquema puede verse también en la figura 4, donde se deduce que, si se baja la ventana virtual una línea hacia abajo cada vez, se irá perdiendo de vista la página 0 e irá apareciendo la página 1 desde abajo hacia arriba.

El efecto de *scroll* de una pantalla es similar al desarrolado por el siguiente algoritmo:

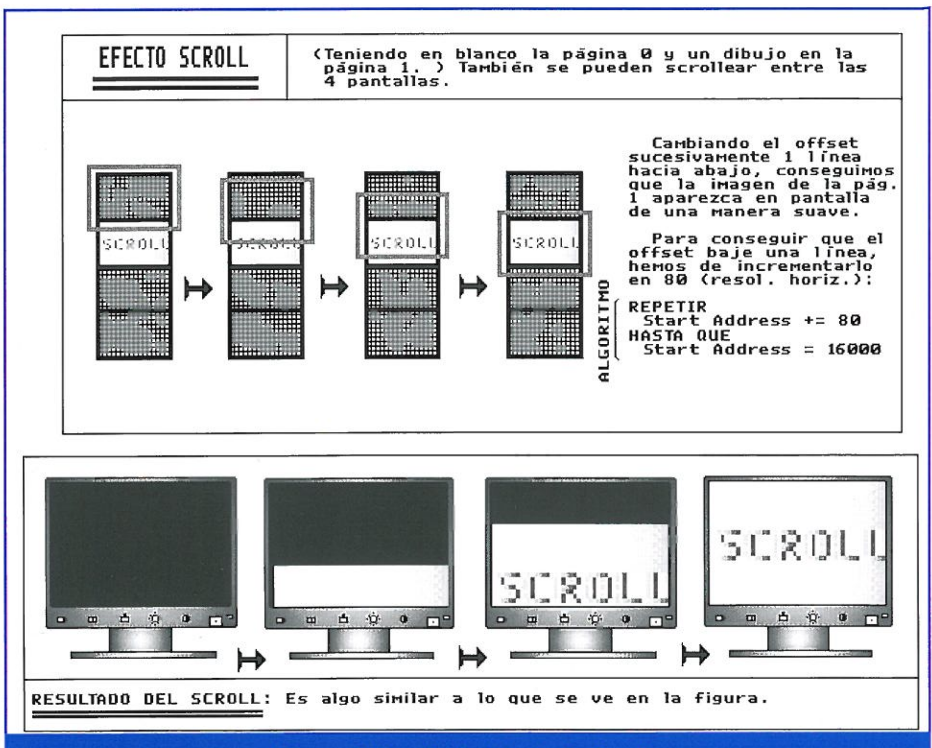


Figura 5. Resultado del efecto Scroll.

1. Dibujar en la página 1 el gráfico a scrollear.
2. Borrar la página 0.
3. Cambiar el *Start Address Register* para que apunte una línea más abajo que donde estaba antes.
4. Esperar un retrazo vertical para ralentizar el efecto.
5. Si aún no se ha alcanzado el principio de la página 1, saltar al paso 3.

Este algoritmo hace que durante su primera ejecución se vean en pantalla las 200 líneas de la página 0 y 0 líneas de la página 1. Al avanzar una línea hacia abajo el marco virtual, se visualizarán 199 líneas de la página 0 y 1 de la página 1, y así sucesivamente.

LA BASE DEL SCROLL

La base del *scroll* consiste el cambio del *Start Address* para colocarlo cada vez una línea más abajo. La librería VGAREGS.H dispone ahora de nuevas funciones de control específicas para el *Start Address*. La primera de estas funciones, *SetAddress()*, modifica el valor de *Start Address* dándole el del argumento que se le pasa. La segunda, *SetPage()*, se basa en la función anterior para facilitar la selección de página usando números de página en lugar de *offsets* en memoria, considerando 0 como la página inicial y 3 como la última página en memoria.

Para realizar el *scroll* es necesario el uso de la función *SetAddress()* para avanzar el "puntero" virtual en memoria en 80 unidades, que es justo la resolución horizontal que tienen los modos *unchained* que se están estudiando (320 píxels/4 planos = 80 píxels por plano).

Teniendo una imagen situada en la página 1 y teniendo la página 0 vacía, veamos el código C que realizaría un *scroll* vertical de una sola pantalla:

```
for( f=0; f<200; f++ )
{
 SetAddress( (f*80) );
  WaitVRetrace();
}
```

El resultado de este código sería algo parecido a lo que puede verse en la figura 5.

Resulta necesario el retrazo vertical ya que sin éste el efecto resultaría demasiado rápido para poder verse. Por supuesto, no hay que limitarse a scrollear sólo una pantalla. Existe la posibilidad de realizar *scrolles* a cuatro pantallas, simular "rebotes" de la pantalla de arriba a abajo ,etc....

CÓMO SUSCRIBIRSE A



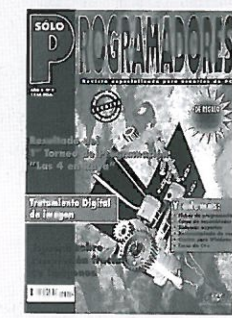



























ríbase enviando este cupón por correo o fax (91) 661.43.86, o llamando al teléfono (91) 661.42.11 Horario 9 a 14 y 15:00 a 18:00 h.

eo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

ΊTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.

ıbre y apellidos...Domicilio..

lación...C.P ....Provincia.........................Telf...............................Profesión...................

ɌMA DE PAGO:

Con cargo a mi tarjeta VISA nº ____ ____ ____ ____

Fecha de caducidad de la tarjeta................Nombre del titular, si es distinto...

Domiciliación bancaria.

Señor Director del banco ....

Población ....

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta

ıorro número..

bo que le será presentado por TOWER COMMUNICATIONS, S.R.L.

pago de mi suscripción a la revista SÓLO PROGRAMADORES.

CODIGO CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

Contra-reembolso del importe más gastos de envío.

Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

Giro Postal (adjunto fotocopia del resguardo).

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Aragoneses, 7
28100 Pol. Ind. ALCOBENDAS (Madrid)

Linux PC

LA LIBRERÍA CURSES

David Aparicio

Algunas aplicaciones no necesitan operar en modo gráfico, bien por sencillez de programación, bien por requerir pocos recursos de sistema. En cualquier caso, la presentación en pantalla de texto también puede requerir algún tipo de vistosidad, como manejo de contextos, posicionado, colores y atributos. Existe una librería que permite realizar estas funciones, y es estándar en Unix.

Unix no está orientado a ser un sistema operativo y nada más. Desde siempre, se ha necesitado organizar la pantalla de texto para conseguir un formateo profesional, con un aspecto similar a aquella vieja guardia de programas de DOS que se han popularizado desde siempre: aplicaciones Clipper, contabilidades, utilidades de gestión y tantas otras. Hasta la llegada de la moda Windows, estas aplicaciones se caracterizaban por un bajo consumo de recursos (2 Mbytes de memoria era casi un lujo) y un desarrollo más o menos rápido, al menos en cuanto a la gestión de entrada/salida. Con la popularidad de los entornos gráficos, las "feas" pantallas de texto han caído en desuso. Sin embargo, el coste en recursos y tiempo de aprendizaje hace que las nuevas aplicaciones no sean todo ventajas.

En cualquier caso, Unix ofrece un entorno estándar para desarrollar programas que utilicen algo más que simples *printf*, y que puedan recompilarse sin sorpresas entre diferentes plataformas. Por supuesto, en Linux se puede encontrar esta librería estándar, entre otros recubrimientos, para hacer más vistosos los programas.

ORGANIZACIÓN

En primer lugar, se puede hacer un pequeño repaso de conceptos para aprender a manejar esta librería. El tratamiento de terminales en Unix tiene varios modos de funcionamiento, de los que se distinguen dos básicos. El modo crudo (*raw*) trata las entradas y salidas sin ningún procesado, sin almacenarlas en búffer intermedio, y sin producir eco de pantalla para las entradas de teclado. El modo procesado (*cooked*) interpreta ciertas secuencias de control, tiene concepto de línea (sólo devuelve control cuando se pulsa el retorno de carro) y realiza un eco de la entrada. Entre los dos modos, existen intermedios que cada programa utiliza según conviene. En los programas clásicos que utilizan el modo normal de entrada salida (*printf/scanf*), el modo de funcionamiento es el procesado. Se puede modificar este comportamiento mediante una familia de las llamadas más complicadas en Unix: el grupo *termio*.

Junto a esta llamada, que puede controlar todos los aspectos del terminal Unix, se conjuga la base de datos que parametriza al conjunto de terminales reconocidos, *terminfo*. Si el terminal empleado para conectar al sistema está reconocido en esta base de datos, y la variable de entorno *TERM* apunta a la entrada correcta, la aplicación podrá visualizarse correctamente en la pantalla.

De cualquier forma, el lector puede evaluar la complejidad del manejo de este aspecto de Unix. Por ello, se ha construido una librería que abstrae al usuario y al programador de la gestión de terminales. Dicha librería, *curses*, es el objetivo de estudio de hoy. Los ingredientes necesarios se reducen entonces a:

- Un terminal reconocido por Linux. En las últimas versiones de *terminfo* se encuentran contemplados casi todos los terminales estándar de hoy. Se instala desde Slackware, y ya está.
- La variable de entorno *TERM* correctamente establecida. Contiene la denominación para *terminfo* del tipo de terminal con que se ha conectado. Programas como *tset* (ver */etc/profile*) detectan automáticamente el tipo de terminal que

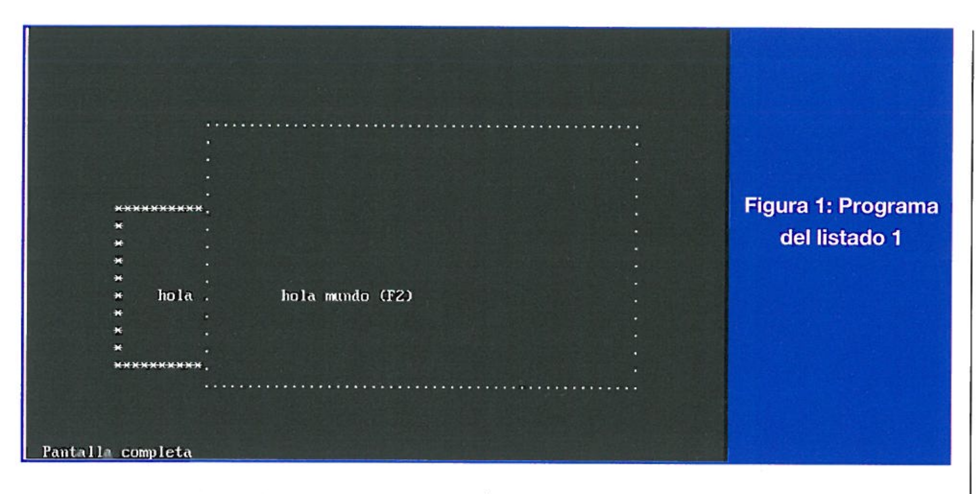


Figura 1: Programa del listado 1

está siendo utilizada, y establecen la variable de forma automática.

- Un programa compilado con el estándar para el formateo de texto a pantalla completa: la librería *curses* y sus derivados.

Con estos tres componentes correctamente establecidos, el programa se mostrará en pantalla adecuadamente.

ESTRUCTURA BÁSICA DEL PROGRAMA

El recubrimiento que se ofrece hace que, desde el punto de vista de la programación, un programa de este tipo es muy similar a uno normal.

En primer lugar, se debe inicializar el estado del terminal y de las estructuras internas, mediante la función *initscr*. Su complementaria, *endwin*, permitirá liberar dichas estructuras de forma organizada cuando finalice nuestro programa. Otra función disponible en la inicialización, tras *initscr*, es *longname*, que proporciona toda la información que pueda necesitar el programa sobre el tipo de terminal en el que se está ejecutando. Esta llamada no suele ser necesaria, puesto que un programa ideal debería ser independiente del tipo de terminal en el que ejecute. Sin embargo, en el "mundo real", unos terminales tienen capacidades de las que otros no disponen, de forma que se puede necesitar el conocimiento de estos aspectos del sistema.

Adicionalmente, cuando se usa esta librería, no se deben llamar a las funciones tradicionales *printf/scanf* y asociadas, sino que la librería proporciona sus propias llamadas. En este momento, estamos en disposición de ver el aspecto del clásico programa de "Hello world":

```
#include <curses.h>

void main(void) {
 initscr();

 mvprintw(10,8,"hola mundo\n");
 refresh();
 endwin();
}
```

Aquí se pueden observar algunos "fenómenos extraños" que se comentan a continuación. Lo primero que puede llamar la atención es la llamada *mvprintw*. Básicamente, se trata de ofrecer la misma funcionalidad que "*printf*" (*print file*), pero adaptándola al nuevo entorno, con "*printw*" (*print window*). En el ejemplo se ha mostrado una variedad de la llamada, que permite conjugar la impresión con el posicionado del cursor. Más adelante se verán estas variedades.

Las aplicaciones de texto son menos vistosas que en modo gráfico, pero requieren menos recursos para ejecutarse

LISTADO 1

```
#include <curses.h>

void main(void) {
 WINDOW *s1,*s2;
 int end;

 initscr();

 mvprintw(24,2,"Pantalla completa");
 wrefresh(stdscr);

 s1 = newwin(10,40,10,10);
 box(s1,'*','*');
 mvwprintw(s1,5,5,"Ventana 1");
 wrefresh(s1);

 s2 = newwin(16,50,5,20);
 box(s2,'.','.');
 mvwprintw(s2,10,9,"Ventana 2");
 wrefresh(s2);

 raw(); noecho();
 while((end=getch()) != 'q')
  switch(end) {
   case '1':
    touchwin(s1);
    wrefresh(s1);
    break;
   case '2':
    touchwin(s2);
    wrefresh(s2);
    break;
   default:
    mvwprintw(stdscr,24,40,
     "(%c) [%3i] [0x%04X]",
     end<' '?'.':end,end,end);
    wrefresh(stdscr);
    break;
  }
 noraw(); echo();
 endwin();
}
```

Un programa curses que utiliza dos ventanas.

El segundo aspecto que llama la atención es *refresh*. Esta función muestra la filosofía básica de la librería. Cuando se utiliza una llamada cualquiera de visualización, en realidad se están manejando unas estructuras internas en memoria. Una vez completado un grupo de llamadas, y cuando el programador decida mostrar los resultados, la llamada *refresh* observa las diferencias entre el buffer interno y la pantalla, y actualiza esta última. Aunque este comportamiento puede no resultar muy interesante en la consola de Linux, donde el tiempo de acceso a las modernas tarjetas VGA PCI es muy similar al de la memoria del sistema, hay que tener en cuenta que la librería trata indistintamente a estos periféricos rápidos o a terminales conectados por línea serie. En este último caso, el optimizar las funciones de redibujado puede mejo-

LISTADO 2

```
#include <ncurses.h>

void main(void) {
 WINDOW *s1,*s2;
 int end;

 initscr(); start_color();
 init_pair(1,COLOR_BLUE,
         COLOR_BLACK);
 init_pair(2,COLOR_GREEN,
         COLOR_BLACK);

 mvprintw(24,2,"Pantalla completa");
 wrefresh(stdscr);

 s1 = newwin(10,40,10,10);
 wattrset(s1,COLOR_PAIR(1));
 box(s1,0,0);
 mvwprintw(s1,5,5,"Ventana 1");
 wrefresh(s1);

 s2 = newwin(16,50,5,20);
 wattrset(s2,COLOR_PAIR(2));
 box(s2,0,0);
 mvwprintw(s2,10,9,"Ventana 2");
 wrefresh(s2);

 attrset(A_NORMAL);
 raw(); noecho();
 while((end=getch()) != 'q')
   switch(end) {
     case '1':
       touchwin(s1);
       wrefresh(s1);
       break;
     case '2':
       touchwin(s2);
       wrefresh(s2);
       break;
     default:
       mvwprintw(stdscr,24,40,
         "(%c) [%3i] [0x%04X]",
         end<' '?'.':end,end,end);
       wrefresh(stdscr);
       break;
   }
 noraw(); echo();
 endwin();
}
```

Un programa ncurses similar al del listado 1.

rar notablemente la velocidad de la aplicación.

En tercer lugar, la llamada a la conclusión del programa, con *endwin*, implica un parámetro que se verá más adelante, *stdscr*, el cual expresa la pantalla completa de la consola que se está tratando, de forma similar al grupo *stdin/stdout/stderr* que representa el dispositivo estándar de los programas normales.

ENTORNOS MULTIVENTANA

Una vez comprobado el sencillo funcionamiento de este sistema, han aparecido algunos indicios que introducen el siguiente apartado. Por un lado, *printw* hace referencia a la impresión sobre una ventana, y por otro, *endwin* utiliza un decriptor de ventana, en este caso el de la pantalla completa. Si el lector se está preguntando si esto significa lo que parece, la respuesta es sí y no.

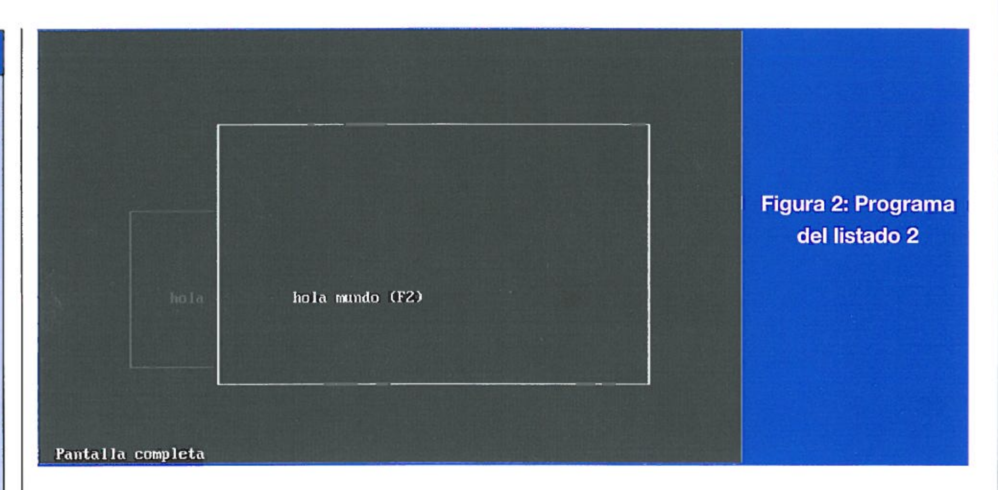


Figura 2: Programa del listado 2

Efectivamente, *curses* tiene un concepto de ventana que permite imprimir en diferentes zonas de pantalla, manteniendo un concepto de identidad independiente entre ellas. Por otro lado, el aspecto negativo es que este manejo no es tan excepcional como se puede pensar, al menos si se utiliza sólo esta librería. No hay llamadas que permitan mover ventanas por pantalla, ni cambiar su geometría, ni iconizarlas, ni tener título, ni botones.

Tras esta cruda enumeración de lo que no está disponible, se pasa ahora a decir lo que sí lo está. Recordando el concepto de la funcionalidad que se ofrece, *curses* permite utilizar una estructura orientada al clásico editor de textos. Es decir, la aplicación nativa que se puede construir imprime textos que se posicionan en pantalla completa, y maneja una serie de ventanas de *pop-up*. Así se permite desplegar diferentes menús e informativos, manteniendo la parte de pantalla que ocultan anotada en algún lugar, de forma que cuando se cierra la ventana, un redibujado del texto recupera los datos y texto originales. Las ventanas solapadas pueden activarse o no, y se permite marcar los bordes con caracteres especiales.

Al evaluar todo esto, se comprenderá que la situación no es mala ni mucho menos, sino que se trata de pensar adecuadamente en la forma de diseñar la aplicación. En el listado 1 se observa un pequeño programa que imprime en dos ventanas con borde y en la pantalla completa, y donde se puede activar una de las dos ventanas mediante las teclas '1' y '2'. Se sale del programa con 'q'.

Aquí aparecen nuevas llamadas al sistema que se pueden comentar. Por una parte, la inicialización del programa es comprensiblemente más elaborada. Las llamadas a *newwin* y *box* crean un nuevo marco en pantalla, de una geometría concreta, y marcan su borde para que el usuario los pueda distinguir.

En segundo lugar, la llamada *refresh* ha sido sustituída por *wrefresh*, que permite especificar la zona de pantalla que se desea actualizar. El programador debe imaginar que *newwin* equivale a *fopen* en los programas convencionales, desde el punto de vista de crear un descriptor nuevo que luego será inmediatamente utilizado en el resto de llamadas de la librería. El descriptor *stdscr* está disponible desde el momento posterior a la llamada a *initscr*, y es un descriptor que representa a "la ventana estándar", paralelamente al concepto de "entrada y salida estándar" de otros programas.

La librería "curses" permite utilizar un terminal a pantalla completa sin conocer el hardware en que se basa

Las primitivas *raw* y *noecho* se usan para modificar el comportamiento del





teclado en el sistema. Por defecto, se asume el comportamiento "elaborado" del terminal. Esto querría decir, por una parte, que sólo tras la tecla de retorno de carro se leerían los caracteres que se hayan pulsado, y algunos serían interpretados en el sistema, en lugar de pasar a la aplicación. Por otra parte, las pulsaciones de teclas se reflejan en pantalla (justo tras el último mensaje que se haya mostrado) produciendo un efecto indeseable en la aplicación. Mediante la invocación de esta pareja, se dispone de una entrada de teclado similar al modo "crudo", mucho más cercano a las necesidades para aplicaciones de ventana como éstas.

La función *touchwin* provoca que la ventana representada por el descriptor referenciado pase a estar en primer plano, tapando la información de otras ventanas solapadas con ella. Si el lector compila y ejecuta este programa, podrá observar el efecto descrito.

Para compilar programas que usen la librería *curses*, se debe pasar a ésta como parámetro, así:

cc -o miexe miprog.c -lcurses

FUNCIONES DE SALIDA

En un programa C estándar, las llamadas para sacar texto por pantalla se dividen en dos grupos básicos: con y sin formateo. Al primer grupo pertenecen *fputc* y *fputs*, y en el segundo se encuentra *fprintf*. Pues bien, en *curses* ocurre algo parecido. Las funciones vistas en los dos primeros ejemplos se basan en la salida formateada, y se agrupan en la siguiente familia:

- *printw(formato, argumentos):* Equivalente a la función *printf*, imprime el texto que se indique en pantalla completa, según el descriptor *stdscr*.
- *wprintw(ventana, formato, argumentos):* Imprime el texto formateado en la ventana indicada.
- *mvprintw(y,x, formato, argumentos):* Como *printw*, escribiendo el texto en la posición indicada en los dos primeros argumentos.
- *mvwprintw(ventana, y,x, formato, argumentos):* La más completa, especifica la ventana y posición relativa dentro de la misma donde escribir el texto formateado.

Junto a estas llamadas, se encuentran las primas de la llamada *scanf*:

- *scanw(formato, argumentos)*
- *wscanw(ventana, formato, argumentos)*
- *mvscanw(y,x, formato, argumentos)*
- *mvwscanw(ventana, y,x, formato, argumentos)*

El cursor también se puede posicionar sin necesidad de escribir o leer información:

- *move(y,x)*
- *wmove(ventana, y,x)*

Para escribir texto sin formatear, al estilo de *puts* y *putc*:

- *addch(char)* y *waddch(ventana, char):* Muestra un carácter en la posición actual de pantalla completa, o de la ventana especificada..
- *mvaddch(y,x, char)* y *mvwaddch (ventana,y,x,char):* Parecidas a las anteriores, pero especificando la posición donde se escribe el carácter.

- *addstr(cadena)* y *waddstr(ventana, cadena):* Muestra una cadena C en la posición actual de pantalla completa, o de la ventana especificada.
- *mvaddstr(y,x, cadena)* y *mvwaddstr(ventana, y,x, cadena):* Como la anterior, pero especificando la posición del cursor.

Existen llamadas para manejar ventanas, inserción y borrado, manejo de atributos y posicionado de texto

Existen también equivalentes para obtener caracteres o líneas completas. Para no hacer monótona la enumeración, simplemente se comenta que equivalen a la familia *add*, sustituyendo estos tres caractéres por *get* en cualquier llamada. Es decir, para la llamada *waddch*, hay una equivalente *wgetch*, con los mismos parámetros.

También se pueden controlar los atributos de los caracteres que se imprimen a continuación, con las siguientes tres parejas de llamadas:

- *attron(atributos)* y *wattron(ventana, atributos):* Activa los atributos indicados en el parámetro.
- *attroff(atributos)* y *wattroff(ventana, atributos):* Desactiva los atributos indicados anteriormente.
- *attrset(atributos)* y *wattrset(ventana, atributos):* Establece los atributos a un valor fijo.

Los valores más usuales que pueden tomar los atributos, siempre que el terminal lo permita, son:

- *A_UNDERLINE*: Subrayado
- *A_REVERSE*: Video inverso
- *A_BLINKING*: Parpadeo
- *A_BOLD*: Intenso
- *A_PROTECT*: No mostrar (password)
- *A_NORMAL*: Valores estándar

Un ejemplo de uso de estas llamadas es:

```
attrset(A_BOLD|A_REVERSE);
printw("Video resaltado+inverso ");
attroff(A_BOLD);
printw("Video inverso ");
attron(A_REVERSE);
printw("Video inverso ");
attrset(A_NORMAL);
printw("Video normal");
```

Las ventanas que *curses* maneja tienen concepto de scroll, y de inserción y borrado de caracteres, mediante las siguientes funciones:

- *insch(char)* y *winsch(ventana, char):* Inserta un carácter en la posición actual, desplazando toda la línea sobrante una posición a la

derecha. El último carácter se pierde.

- *mvinsch(y,x,char)* y *mvwinsch(win, y,x, char):* Similar a los anteriores, pero en una posición concreta de la pantalla o ventana.
- *insertln()* y *winsertln(ventana):* Inserta una nueva línea en la posición actual, desplazando el resto de ellas hacia abajo. La última se pierde. Si se realiza esta función sobre la primera línea de la pantalla o ventana, se produce un scroll hacia abajo de todo el área, sin intervención del hardware del terminal.
- *scroll(ventana):* Desplaza todo el área indicada una línea hacia arriba, con intervención del hardware si se trata de la pantalla completa.
- *scrollok(ventana, flag):* Activa o desactiva la capacidad automática de una ventana para producir un scroll cuando se escribe pasada la esquina inferior derecha de la misma.
- *delch()* y *wdelch(ventana):* Borra el caracter de la posición actual, desplazando el resto de la línea una posición a la izquierda, y rellenando con un blanco el último carácter.
- *deleteln()* y *wdeleteln(ventana):* Borra la línea actual, produciendo un desplazamiento del resto del área una línea hacia arriba. Si se realiza en la primera línea, se produce un efecto similar a *scroll*, pero por software.

Las funciones para el limpiado de la pantalla son las siguientes:

- *erase()* y *werase(ventana):* Deja toda la ventana en blanco, y establece atributos por defecto para la misma.
- *clear()* y *wclear(ventana):* Similar al anterior, pero además se marca una condición para que se redibuje toda la pantalla cuando se llame a *refresh* o *wrefresh*. Esto permite eliminar posibles defectos de redibujado, y deja la pantalla en un estado conocido.
- *clrtobot()* y *wclrtobot(ventana):* Borra todos los caracteres desde la posición del cursor hasta el margen inferior derecho de la pantalla o ventana especificada.
- *clrtoeol()* y *wclreol(ventana):* Limpia desde la posición del cursor hasta el final de la línea.

Otras llamadas disponibles en la librería son:

- *cbreak()* y *nocbreak():* Activa o desactiva el modo línea (los caracteres tecleados se procesan tras un retorno de carro) o modo carácter a carácter. Los caracteres de control se interpretan en cualquier caso.
- *echo()* y *noecho():* Activa o desactiva el mostrado en pantalla de los caracteres que se teclean.
- *nl()* y *nonl():* Activa o desactiva la conversión entre ls caracteres NL y NL+CR.
- *raw()* y *noraw():* Parecido a *cbreak*, pero los caracteres de control no son interpretados.
- *resetty()* y *savetty():* Salvaguarda y restablece el estado del terminal, cuando se efectúan varias operaciones sobre él, y se desea un valor de referencia.
- *refresh()* y *wrefresh(ventana):* Copia el contenido de los descriptores internos de las ventanas a pantalla.
- *touchwin(ventana):* Marca una ventana para ser totalmente redibujada cuando se llama a *refresh* o *wrefresh*.
- *beep()* y *flash():* Permiten mostrar una condición de error, emitiendo un sonido o produciendo dos inversiones consecutivas de atributos en la pantalla completa.
- *box(ventana, vert, hor):* Permite dibujar un borde en una ventana determinada, especificando el carácter de las líneas verticales (*vert*) y horizontales (*hor*).
- *newwin(lineas, columnas, x,y):* Devuelve un descriptor (de tipo *WINDOW* *) con una nueva ventana de las dimensiones indicadas y en la posición deseada.
- *delwin(ventana):* Destruye una ventana creada con la llamada anterior.
- *initscr():* La primera función invocada en el programa, permite inicializar las estructuras internas de la librería.
- *endwin():* Se invoca al salir de un programa, y restablece las condiciones del terminal antes de invocarlo.

LAS NUEVAS LIBRERÍAS

Lo que se ha visto hasta ahora es la librería clásica para el formateo de pantallas en Unix. Sin embargo, y conforme ha avanzado la capacidad de los terminales para embellecer su visualización (con colores, por ejemplo), las prestaciones de la librería original han oscurecido ciertos aspectos del modelo clásico. Con la llegada de las versiones más modernas de Unix (familia denominada System V), apareció una nueva versión de la librería: *ncurses* (acrónimo por '*new curses*'). Junto a ella, se han añadido otras librerías (no tan extendidas ni estandarizadas, pero muy útiles) que proporcionan nuevos servicios, como el control de menús y diálogos, y que se basan en la capa básica de *curses*.

Todo lo visto hasta ahora se aplica completamente para la nueva librería, que es un superconjunto de la antigua, y, por tanto, es compatible. Sin embargo, a continuación se verán unos ejemplos que enseñan las mejoras visuales que se pueden obtener.

En el listado 2 hay un ejemplo de adaptación del listado 1 (que habría compilado perfectamente con el nuevo sistema) para mostrar la diferencia de aspecto, si se comparan las figuras 1 y 2.

Se pueden ver ciertas diferencias leves entre ambos códigos. Por un lado, los marcos de las ventanas aprovechan el juego de caracteres del PC, cuando se especifica como borde el carácter 0 en la primitiva *box*. Además, el soporte de color extiende la capacidad de las primitivas de atributos (*attrset*, *attron* y

La librería "ncurses" proporciona un aspecto más elaborado que la librería original, y es totalmente compatible

attroff). En esta librería, se tratan los colores en pares, asociando un atributo especial a una combinación de color de fondo y color de letra, mediante la primitiva *init_pair*. Previamente se han iniciado las capacidades de color, con la función *start_color*. Para usar una combinación, se establece un atributo especial, que se contruye con la macro *COLOR_PAIR*, tal y como se observa en el listado 2. Se puede encontrar mayor información de este comportamiento con el comando "man curs_color".

La última diferencia de este ejemplo es el fichero que se incluye como cabecera de librería, que es *ncurses.h*. Para compilar este nuevo ejemplo, también habrá que enlazar el nuestro con la librería adecuada:

cc -o miejem miejem.c -lncurses

El paquete que incluye esta nueva librería (y sus asociadas, que se enumerarán a continuación) se encuentra en la serie D de Slackware 3.0, con el nombre *ncurses.tgz*. Se tiene que instalar este paquete de la forma habitual para poder seguir los ejemplos de esta sección.

En la capacidad original de *curses*, las ventanas que se solapan no guardan los fondos unas de otras (*back store*), sino que se requiere redibujar la ventana vigente en cada momento. Con *ncurses*, las venatnas solapadas guardan contexto del fondo que están tapando, y lo restablecen adecuadamente cuando son destruídas, lo que se traduce en un aumento de la velocidad de dibujado.

Otras características relevantes de la nueva librería son el soporte de menús (librería *menu*) y la capacidad de definir ventanas de mayor tamaño que la pantalla (*pads*). Se puede ver un ejemplo de uso de menús en el software que acompaña a la revista, en el programa *menu.c*.

Las extensiones que acompañan a esta librería se deben enlazar con el ejecutable. Por ejemplo, para usar capacidades de menús, la cabecera *menu.h* debe incluirse en el programa (*ncurses.h* es incluida por la anterior de forma automática), y se compila con:

cc -o miprog miprog.c -lmenu -lncurses

Como se puede observar, las librerías que se apoyan en *ncurses* se deben invocar primero en la línea de comando del compilador, ya que incluyen referencias a la librería general. No hacerlo así produciría un error en el enlazado, puesto que se realiza una sola pasada para resolver las llamadas entre módulos del compilador.

Se puede encontrar información adicional para estas extensiones en el paquete si se expande documentación en el subdirectorio */usr/doc/ncurses*, y fuentes de ejemplo en */usr/src/ncurses*. Un punto de partida también puede ser el siempre fiel manual, con:
man ncurses
man menus

CONCLUSIONES

Unix nació en instalaciones donde un ordenador central ofrecía servicio a terminales remotos, simples pantallas conectadas por línea serie. La necesidad de visualizar correctamente una aplicación a pantalla completa en diferentes modelos de terminales promovió la estandarización de una librería de visualización textual, que escondió las diferencias de hardware entre diferentes equipos.

Con la distribución de la potencia de cálculo en máquinas de sobremesa, el concepto de "pantallas de texto" ha cambiado, puesto que se basan en terminales rápidos y a color. De la librería *curses*, proporcionada con Unix BSD, se ha pasado a la *ncurses* de System V, con capacidad de color, mejora en la representación de las pantallas y más apoyo en las capacidades hardware de los sistemas actuales.

Para los profesionales que desarrollan aplicaciones distribuídas, los entornos gráficos pueden no ser la mejor opción, puesto que requieren máquinas con más requerimientos hardware (quizá para ser meros terminales), y además un conocimiento profundo de la programación de entornos gráficos distribuídos.

En este mercado concreto, los terminales de texto siguen siendo una solución sencilla y efectiva. Si este es el caso, curses es un estándar a tener en cuenta y que, por un lado, facilita la migración entre plataformas, y por otro permite concentrarnos en lo que se desea que el programa haga, y no en cómo debe verse. El día en que MS-Windows sea realmente estable y permita visualizar aplicaciones que corran en una máquina y sean visualizadas en la pantalla de otra, o cuando X-Windows tenga su "Visual Basic" y sea tan sencillo de programar (*Tcl/Tk* todavía no es perfecto), este artículo dejará de tener vigencia. Pero todavía hoy no es ese día. Hasta otra.

Un programa que usa "curses" se puede recompilar en cualquier plataforma Unix sin ser modificado

Existen extensiones para facilitar el manejo de pantalla de los programas, como 'menus' y 'pads

El modo crudo (raw) trata las entradas y salidas sin ningún procesado

COMPRESIÓN POR CÓDIGOS HUFFMAN

Francisco José Abad Cerdá

El tema de la compresión de datos ha tomado gran importancia para la mayoría de los usuarios. Para muchos tiene algo de esotérico (¿cómo meter en un disco de 1.44Mb un programa que ocupa 3Mb?) que se intenta desvelar en éste artículo.

La compresión ha ido tomando gran importancia en la mayoría de los aspectos de la informática en los que interviene un intercambio de datos, ya sea directamente por parte del usuario (¿quién no ha usado un compresor antes de copiar ficheros a disquetes?) o bien ya sea el propio sistema el que lo haga transparentemente (vídeo digital, teleconferencia, transmisión telefónica de datos, etc). Esto ha sido en gran modo debido al incremento espectacular de la capacidad de cálculo de los microprocesadores, en contraposición con el ancho de banda (capacidad de transmisión) de los medios, que ha crecido de forma menos espectacular. Así, usualmente se carga a la CPU con un trabajo extra, para reducir al mínimo el uso de la línea de comunicaciones. Con la compresión no sólo se consigue un ahorro de espacio, sino también un aumento de la velocidad, ya que la descompresión usualmente tomará varios órdenes menos de magnitud que la simple transferencia.

En este artículo se va a introducir el método de compresión de datos inventado en 1952 por el matemático francés Huffman, del que hereda el nombre.

Dentro de la inmensidad de algoritmos de compresión de datos existentes, podemos hallar dos tipos principales: los reversibles (*lossless*) y los irreversibles (*lossy*). La diferencia entre uno y otro está en que mientras los primeros reconstruyen los datos originales tales como eran, sin pérdida de información, los *lossy*, no pueden restaurar los datos originales al cien por cien, pero a cambio alcanzan una mayor compresión. Esto puede llevar a creer que no tienen utilidad alguna, pero un análisis un poco más profundo permite encontrar múltiples aplicaciones para éstos algoritmos. Los algoritmos *lossy* se usan principalmente en la compresión de imágenes digitalizadas, en los que la mayoría de las veces un observador humano no podrá distinguir la imagen original y la comprimida. Ejemplo de ésto es el conocido formato JPEG, que alcanza porcentajes de compresión escandalosos comparado con los formatos tradicionales. En general estos procedimientos se usan en entornos que mueven muchos datos en tiempo real, y no requieren una precisión total.

El procedimiento que se explica en este artículo es del tipo reversible, por lo que es válido para comprimir cualquier tipo de ficheros (no es muy aconsejable usar un algoritmo lossy con archivos ejecutables). Comparado con cualquier compresor "de verdad" (ARJ, PKZIP, etc), el programa que acompaña al artículo parece de juguete, ya que ni se acerca a los porcentajes de compresión de estos programas. En descargo, hay que tener en cuenta que algunos de estos usan los códigos Huffman en alguna fase intermedia de su procesamiento.

Cualquier algoritmo de compresión, en líneas generales se basa en eliminar redundancia de los datos originales. Se considera redundancia como toda aquella información que no es esencial en un conjunto de datos. Hay varios tipos de redundancia, pero la mayoría de los algoritmos intentan sacar partido de una repetición constante en los datos, sustituyendo la cadena repetida por otra cadena de menor tamaño. En ésto se basan los algoritmos de códigos de longitud variable y el LZW. Veamos un simple ejemplo:

Datos originales: AAABBBBCDEA-ABB

Posible compresión: 3A4BCDE2A2B

En el proceso de descompresión, si

Figura 1: Proceso "a la derecha" del algoritmo. Cada columna representa un paso del bucle.

B ⟶ 0.429 ⟶ 0.429 ↘ 0.571
A ⟶ 0.357 ⟶ 0.357 ↗ 0.429
D ⟶ 0.143 ⟶ 0.214
C ⟶ 0.071

se encuentra un número, se lee el siguiente carácter y se replica tantas veces como indique ese número. Si aparece una letra, se copia tal cual. En un procedimiento parecido al anterior se basa el conocido formato de imágenes PCX.

El código Huffman no se aprovecha de las repeticiones de cadenas, sino que intenta recodificar los datos originales de una cierta forma. Debido a sus propiedades, parece adecuado para recoger la salida de otro procedimiento de compresión, tal como códigos de longitud variable, LZW, compresión aritmética, etc.

Para entender un poco las propiedades de la codificación Huffman, hay que acudir a la teoría de la información de Shannon. Esta teoría define un bit como la cantidad de información de un suceso que tiene una probabilidad de ocurrencia del 50 por ciento. El socorrido ejemplo de la moneda lanzada al aire representa exactamente un bit. Si alguien dice "Al lanzar la moneda ha salido cara", ha emitido un bit de información.

Matemáticamente, ésto se expresa así:

$I(E)=\log_2 1/P(E)$

Donde I(E) es la información en bits del suceso E y P(E) es la probabilidad del suceso E.

Supongamos una fuente de información que puede producir sucesos $\{a_1,a_2,...,a_n\}$ y un alfabeto de símbolos $\{b_1,...,b_n\}$. Un código es una correspondencia que asocia sucesos de A a símbolos de B. El ordenador y el usuario están acostumbrados al código ASCII, que no hace más que asignar a un conjunto de sucesos {"A","B","C",...,"1","2",...,[ESC]...} un conjunto de números {65,66,67,...,49,50,...,27...}.

Shannon dice que la longitud media de un código es la media del número de bits que componen una palabra. Por ejemplo:

Los compresores intentan reducir el nivel de redundancia en los datos

Dado un alfabeto de 4 códigos y su probabilidad de aparición:

Código	Probabilidad
A	50%
B	25%
C	15%
D	10%

La codificación normal de los 4 códigos en binario sería:

A -> 00 B -> 01 C -> 10 D -> 11

La longitud media del código anterior es:

$$L = \sum_{i=0}^{4} P_i*L_i=0.5*2+0.25*2+0.15*2+0.1*2=2 \text{ bits/símbolo.}$$

siendo P_i la probabilidad de aparición del código i, y L_i la longitud del código i

Una codificación alternativa, pero no tan evidente puede ser:

A -> 0 B -> 10 C -> 110 D -> 111

$$L = \sum_{i=0}^{4} P_i*L_i=0.5*1+0.25*2+0.15*3+0.1*3=1.75 \text{ bits/símbolo.}$$

con lo que por cada símbolo que se codifica de éste modo, se ahorra un 12.5 por ciento de espacio respecto a la codificación anterior.

En suma, la codificación Huffman busca asignar a los códigos que más se repiten el menor número posible de bits. Se puede demostrar que el procedimiento Huffman consigue un código de longitud media mínima.

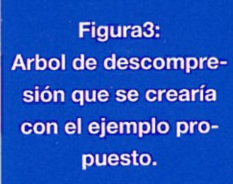
Por lo tanto, los datos comprimidos serán una larga serie de bits, donde no se sabe, a priori, dónde empieza un símbolo u otro, puesto que no hay separadores entre un código y el siguiente y además el tamaño de cada símbolo es distinto. Pero si nos fijamos un poco en el ejemplo anterior, se puede ver que nunca se dará un caso en el que no se sepa qué byte corresponde decodificar a continuación. Por ejemplo:

0011010111010100110111 -> AACBDABACD

Se dice que el código Huffman es un código instantáneo, ya que no necesita memoria auxiliar para su decodificación. Es decir, no necesita ir creando

Figura 2: Proceso "a la izquierda" del algoritmo.

B ⟶ 0.429 [1] ⟵ 0.429 [1] ↖ 0.571 [0]
A ⟶ 0.357 [00] ⟵ 0.357 [00] ↙ 0.429 [1]
D ⟶ 0.143 [010] ⟵ 0.214 [01] ↙
C ⟶ 0.071 [011]



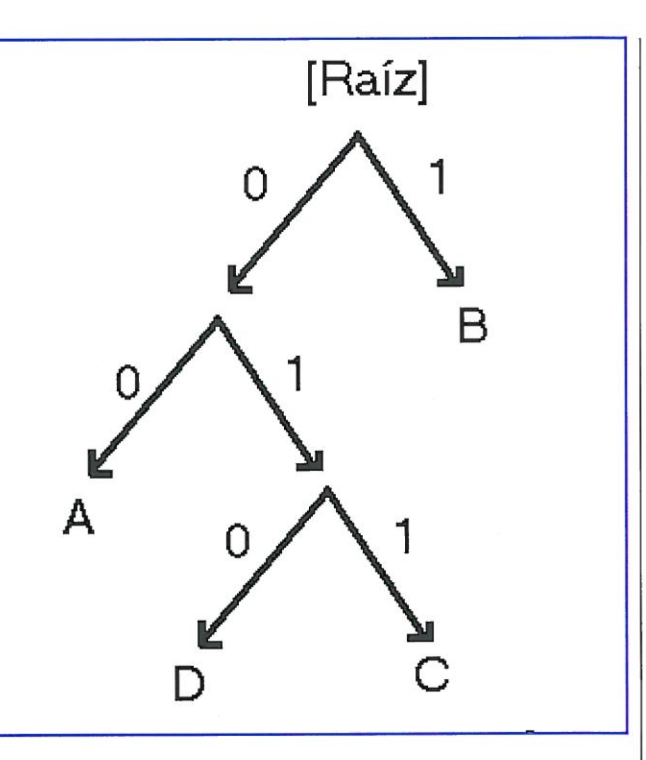


Figura3: Arbol de descompresión que se crearía con el ejemplo propuesto.

tablas auxiliares tal y como necesita hacer, por ejemplo, el procedimiento LZW. Después se intentará aprovechar ésto en la sección de descompresión.

COMPRESIÓN

En el esquema 1 se presenta una implementación en pseudocódigo del procedimiento.

Como se puede apreciar, el algoritmo se divide en tres partes bien diferenciadas: inicialización, primer bucle Mientras y segundo bucle Mientras.

Cuando se hace en papel, se pueden ver claramente dos "movimientos": uno hacia la derecha, que es el correspondiente al primer bucle y que va calculando las frecuencias acumuladas de dos en dos y otro posterior a la izquierda que ya va colocando los bits de cada código.

Por lo tanto, en el primer proceso va decreciendo el número de filas en la tabla cada vez que se suman dos, pero hay que guardar de alguna forma una especie de "punteros hacia atrás" que indiquen qué posición ocupaba una cierta fila en el proceso anterior.

Para clarificar el método, se presenta un ejemplo práctico de una compresión completa:

Datos originales:
ABDBCDBBABABAA

Hallar:
A -> 5/14 -> 0.357
B -> 6/14 -> 0.429
C -> 1/14 -> 0.071
D -> 2/14 -> 0.143

Ordenar histograma:histograma:
B -> 6/14 -> 0.429
A -> 5/14 -> 0.357
D -> 2/14 -> 0.143
C -> 1/14 -> 0.071

En la figura 1 se muestra el proceso "a la derecha" (primer Mientras).

Cuando quedan dos filas, es momento de asignar a una un 0 y a la otra un 1, y empezar el proceso "a la izquierda" (figura 2).

Luego el fichero original quedaría comprimido simplemente sustituyendo cada carácter del fichero original por su correspondiente codificación resultante, del siguiente modo:

0010101011010110010010000

El método es análogo para cualquier número de códigos en el alfabeto (en el caso de un fichero binario cualquiera, tendrá como mucho 256 códigos distintos).

DESCOMPRESIÓN

El caso de la descompresión es mucho más fácil que el de la compresión. La estructura de datos que contiene los símbolos y la codificación correspondiente debe ser almacenada en el proceso de codificación, para poder usarla en ésta etapa. En el ejemplo anterior, sería algo así:

A -> 00 B -> 1 C -> 011 D -> 010

Lo único que resta es ir leyendo la serie de bits que forman el fichero comprimido, e ir sustituyendo cada serie de bits que forme un código por su valor original correspondiente.

Uno de los modos más rápidos para descomprimir de éste modo, y el que se ha implementado en el programa adjunto, es mediante un árbol binario, en el que cada hijo del nodo representa la posibilidad de ir hacia la izquierda (0) o a la derecha (1) según el bit a decodificar. El ejemplo anterior crearía un árbol parecido al de la figura 3.

Así, en cada momento el proceso se encontrará en un determinado nodo del árbol. Si el bit siguiente a decodificar es

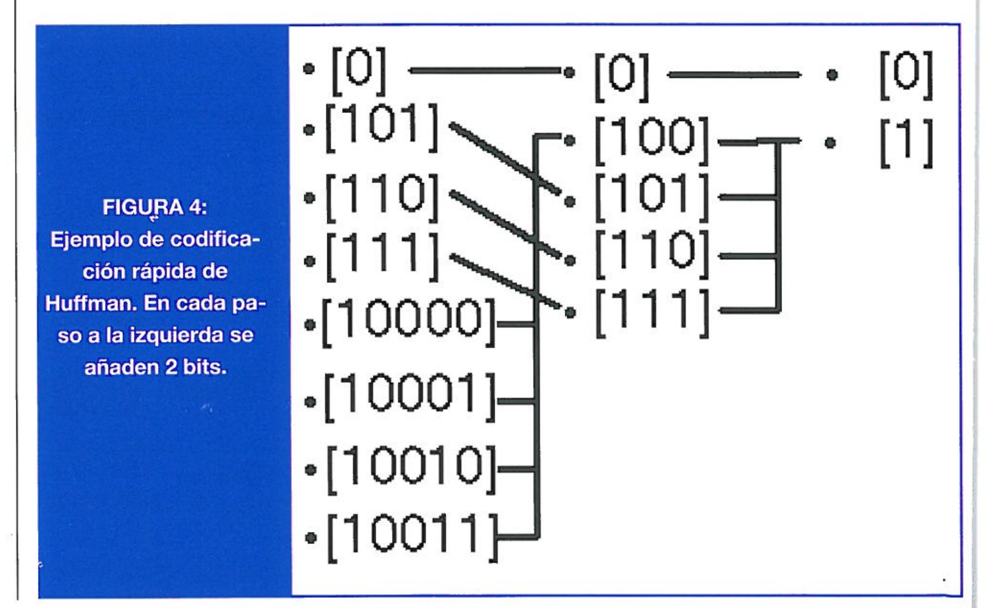


FIGURA 4: Ejemplo de codificación rápida de Huffman. En cada paso a la izquierda se añaden 2 bits.

ESQUEMA 1: Pseudocódigo del algoritmo de compresión de Huffman.

```
Procedimiento Comprimir
Comenzar
   Calcular histograma del fichero
   Ordenar histograma por probabilidad de aparición
   (* Si hay n bytes distintos, habrán n filas *)
   Mientras n>2 hacer
      Sumar probabilidades de dos últimas frecuencias
      Marcar los predecesores de la nueva fila
      Reordenar la tabla por probabilidades
   FinMientras
   (* Quedan dos filas *)
   Asignar a una fila un 0 y a la otra un 1
   (* Formar los códigos siguiendo los punteros hacia atrás *)
   Mientras queden filas con predecesores hacer
      Desde i=0 hasta n hacer
                        (* n número de filas en la tabla actual *)
         Si fila[i] tiene un predecesor entonces
            Asignar al predecesor el código de fila[i]
         sino   (* tiene dos predecesores *)
            comenzar
               Copiar código de fila[i] a predecesores
               Añadir un 0 a un predecesor y 1 al otro
            fin
         FinSi
      FinDesde
   FinMientras
Fin Comprimir.
```

un 0, se baja por el nodo izquierdo. Si el nodo contiene un símbolo, se manda al fichero de salida y se vuelve a la raíz del árbol. Si el bit de entrada fuera un uno, el proceso seguiría por la derecha análogamente. Aquí se puede apreciar lo que se anunciaba al principio: en cada momento, sólo importa el siguiente bit a decodificar, pudiendo olvidar los bits pasados.

Los compresores comerciales usan en alguna etapa intermedia los códigos Huffman

VARIANTES Y APLICACIONES

Como ya se ha dicho, los compresores comerciales usan en algún paso intermedio la codificación Huffman, pero sus utilidades no acaban ahí. El estándar de la CCITT relativo al FAX usa una variante de códigos de Huffman, en la que los códigos ya están precalculados para una serie de documentos estándar y que guarda el fax en una memoria ROM.

El inconveniente de la codificación Huffman, como se puede suponer, es la lentitud a la hora de comprimir datos. Para solucionar esto se diseñó la codificación rápida de Huffman que, en vez de coger filas de dos en dos, las coge de 4 en 4, o de 8 en 8. En vez de añadir cada vez un bit en el paso hacia la izquierda, se añaden 2 o 3. Este procedimiento necesita mucho menos cálculo que el original, pero consigue porcentajes de compresión peores que el Huffman original. En la figura 4 aparece un ejemplo de la codificación rápida de Huffman tomando cada vez 4 filas y añadiendo dos bits en vez de uno.

Otra variante de la codificación Huffman son los códigos de Huffman truncados, que son aquellos en los que sólo se codifica una parte de los datos, mientras que la otra se codifica en binario. Esto sería útil, por ejemplo si se conoce la naturaleza y la disposición de los datos a comprimir. Se comprimiría la parte que contuviera más bytes repetidos, dejando la parte más heterogénea sin comprimir.

ESQUEMA 2: Estructura general de la función Huffman_rec.

```
Procedimiento Huffman_rec( tabla *p, int cuantos)
 Comienzo
  Si cuantos>2 entonces
    Comienzo
     Crear una nueva tabla (mitabla)
           Calcular la suma de las frecuencias de las dos últimas filas
           Ordenar la nueva tabla
             (* Se calcula recursivamente la solución a la tabla *)
           Huffman_rec(mitabla, cuantos-1);
             (* En éste momento, la tabla global "código" contiene el proceso de construcción de los
códigos desde el final a mitabla. Ahora se debe reorganizar la tabla códigos de acuerdo con los
punteros hacia atrás y añadir los bits correspondientes de desdoblar
la suma de las dos últimas filas *)
             (* Acaba la recursión en mitabla *)
           Liberar(mitabla)
           Fin
   sino (* cuantos=2 *)
       (* Este es el primer paso del proceso "a la izquierda".*)
           Marcar las dos únicas filas que hay como 0 y como 1.
Fin Huffman_rec.
```

IMPLEMENTACION

La única dificultad que presenta el programa incluido es la función que calcula los códigos Huffman y que debe hacer el recorrido en los dos sentidos. En el caso del programa, se llama Huffman_rec, y se le pasa un puntero a una tabla que contiene los valores de frecuencias acumuladas, los punteros hacia atrás y el número de filas en la tabla. Un esquema general podría ser el que aparece en la esquema 2.

El programa adjunto está documentado y explica la implementación hecha paso a paso. Como se puede apreciar, el programa hace uso intensivo de la recursividad, y como se enseña en cualquier curso de primero de programación, la recursividad es cara, pero en éste caso se simplifica mucho el proceso de "ir y volver". Una de las posibles optimizaciones del algoritmo sería transformar el proceso recursivo en uno iterativo, pero si el código recursivo es ya poco legible, el iterativo puede ser un verdadero jeroglífico.

Una de las cosas que debe hacer cualquier programador que ponga la vista en acelerar al máximo sus programas es evitar manejar números en coma flotante, ya que ralentizan el proceso de cálculo enormemente, y no sólo en aquellos ordenadores sin coprocesador matemático. Además, como el programador no sabe en qué ordenador se ejecutará su programa, debe incluir las librerías matemáticas, que permiten a los ordenadores sin coprocesador usar números reales, emulando a éste dispositivo, pero lo cual "engorda" innecesariamente el programa para aquellos ordenadores que sí lo poseen. Se pueden usar varios trucos para trabajar con enteros pero dándoles algunas cifras decimales, lo que basta para la mayoría de los casos. Esto se hace normalmente jugando con desplazamientos que funcionan exactamente como divisiones y multiplicaciones por potencias de dos.

Hay variantes al algoritmo original de Huffman que incrementan la velocidad

Pero en el caso del programa ésto es tan sencillo como crear el histograma no con la probabilidad de aparición de un cierto dato (un número entre 0 y 1), que es como se hace por definición el histograma, sino con el número de veces que aparece el dato. Con un ejemplo ésto se verá más claro:

h_i=probabilidad de aparición del dato i

$$Sh_i=1$$

Cada h_i es n_i/N, siendo n_i el número de apariciones del dato i, y N el número de datos

Por lo tanto:

$$Sn_i/N=1$$

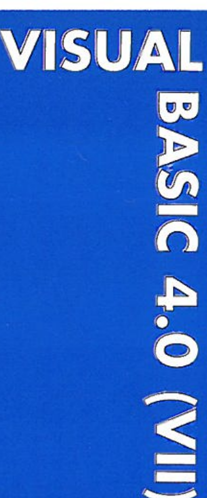
Multiplicando por N:

$$Sn_i=N$$

Como para lo único que se quiere el histograma es para ordenar los datos por orden de aparición, da lo mismo trabajar con probabilidades que con número de apariciones, pero nos ahorramos los costosos cálculos en coma flotante. Este ejemplo puede parecer un poco burdo, pero los procedimientos que hagan un uso intensivo de cálculos matemáticos incrementarán notablemente su velocidad si se incorporan éstos "trucos".

La descompresión se puede hacer rápidamente si se usa un árbol

Una curiosidad: El programa está preparado para ser compilado en Linux, siendo totalmente compatible a nivel de archivos comprimidos entre DOS y Linux. Además ha sido probado en una máquina AVIION corriendo un sistema Unix DG/UX y sobre una Sun con muy pocos cambios.



CONCEPTOS DE PROGRAMACIÓN

Juan Manuel y Luis Martín

Las aplicaciones realizadas con Visual Basic se crean a partir de proyectos compuestos por módulos. Según el nivel de complejidad de la aplicación, serán necesarios más módulos para llevar a cabo las distintas tareas. Visual Basic dispone de tres tipos de módulos:

- Módulos de formulario: Contienen la interfaz visual de las distintas ventanas de la aplicación, así como el código necesario para su funcionamiento. Por tanto, especifican las propiedades y procedimientos de evento tanto del formulario como de los controles que contiene.
- Módulos estándar: Contienen solamente código en forma de procedimientos, que pueden ser compartidos por otros módulos de la aplicación.
- Módulos de clase: Contienen código con la definición de una clase a partir de la cual pueden crearse objetos. Por tanto, especifican las propiedades y métodos que caracterizan a los objetos de dicha clase.

Cada uno de los tipos de módulos anteriores se componen de tres partes:

- Declaraciones: Código no ejecutable que permite definir constantes, variables y tipos de datos, así como especificar sus características. También permite declarar los procedimientos *DLL* que van a ser utilizados.
- Procedimientos de Evento: Bloques de código ejecutable que realizan tareas específicas y cuyo código se ejecuta como respuesta a un evento. Sólo se incluyen en los módulos de formulario.
- Procedimientos Generales: Bloques de código ejecutable que realizan tareas específicas y cuyo código se ejecuta por invocación directa desde el código de otro procedimiento.

Tanto las declaraciones como los procedimientos se realizan mediante la escritura de líneas de código secuenciales, que representan instrucciones concretas para el ordenador.

INSTRUCCIONES Y LÍNEAS DE CÓDIGO

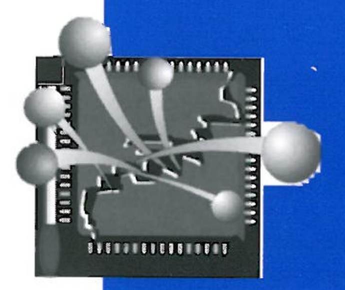
Las instrucciones se forman mediante la combinación de palabras reservadas de Visual Basic con expresiones del lenguaje, entendiendo estas últimas como conjuntos de variables y constantes relacionadas entre sí mediante operadores. En general, puede decirse que una instrucción comienza por una sentencia, que es la palabra reservada que indica el tipo de acción a realizar.

```
Load Form1
MsgBox "Hola"
Set Obj = Text1
```

La única excepción la constituyen las sentencias de asignación y las llamadas a procedimiento que, aunque provienen de las sentencias *Let* y *Call*, es posible omitirlas.

```
x = 0 'Equivale a: Let x = 10
Calcular 'Equivale a: Call Calcular
```

Cada línea de código de un procedimiento contiene una instrucción. Sin embargo, es posible incluir más de una instrucción, separándolas entre sí mediante el carácter dos puntos ":".



La realización de una aplicación con Visual Basic puede dividirse en varias etapas, según se vio en artículos anteriores. De entre ellas, la etapa de elaboración del código es quizás la que requiere un mayor nivel de conocimientos por parte del programador, ya que son necesarios ciertos conocimientos del lenguaje de programación. En este número se describen algunos conceptos básicos de la programación con Visual Basic, así como los tipos de datos, sus operadores y funciones de tratamiento.

Instrucción1 : Instrucción2 : ... : InstrucciónN

También es posible escribir una instrucción que ocupe mas de una línea de código, utilizando para ello el carácter de continuación "_".

```
Image1.Move x + 10,_
        y + 10
```

Como en la mayoría de los lenguajes de programación, es posible insertar comentarios en el código, utilizando el carácter comilla simple "'". De esta forma, el resto de la línea se interpreta como un comentario.

```
'Esto es un comentario
Sub Command1_Click()  'Esto también
```

TIPOS DE DATOS

Visual Basic trabaja con datos de distinta naturaleza (números, texto, fechas, etc). Para ello dispone de dos tipos de datos primarios, a partir de los cuales se derivan otros. Se trata de los Números, datos de tipo aritmético compuestos por cifras, y las Cadenas o *strings,* datos de tipo texto compuestos de series de caracteres.

Los datos pueden almacenarse en variables y propiedades, o bien indicarse como literales. En este último caso, los números se expresan normalmente, mientras que las cadenas deben ir encerradas entre comillas. Además de en decimal, es posible indicar números en formato hexadecimal y octal, utilizando para ello los prefijos *&H* y *&O*, respectivamente.

```
"Esto es una cadena"
125.43  'El punto se utiliza como separador decimal
&H3F    '63 en formato hexadecimal
&O137   '95 en formato octal
```

En el caso de las variables, es necesario declararlas con la sentencia *Dim*, cuya sintaxis es la siguiente:

```
Dim Variable [As Tipo],...
```

TABLA 1

Tipo de datos	Sufijo	Tamaño	Rango de valores
Byte		1 byte	De 0 a 255
Boolean		2 bytes	True o False
Integer	%	2 bytes	De -32,768 a 32,767
Long	&	4 bytes	De -2,147,483,648 a 2,147,483,647
Single	!	4 bytes	De -3.402823E38 a -1.401298E-45 para negativos, y de 1.401298E-45 a 3.402823E38 para positivos.
Double	#	8 bytes	De -1.79769313486232E308 a -4.94065645841247E-324 Para valores negativos, y de 4.94065645841247E-324 a 1.79769313486232E308 para positivos
Currency	@	8 bytes	De -922,337,203,685,477.5808 a 922,337,203,685,477.5807.
Date		8 bytes	Del 1 de Enero del 100 al 31 de Diciembre del 9999
Object		4 bytes	Referencias a objetos
String	$	Variable	Con tamaño variable 10 bytes + 1 byte/carácter. Con tamaño fijo, 1 byte/carácter.(Hasta 65.400 caracteres)
Variant		Variable	Con números, 16 bytes. Con caracteres, 22 bytes + 1 byte/carácter
Type		Variable	El rango de cada tipo fundamental

Tipos de datos fundamentales de Visual Basic.

La cláusula *As Tipo* permite indicar el tipo de datos que contendrá la variable. Sin embargo, cuando se omite esta cláusula, Visual Basic declara la variable como de tipo *Variant.* Se trata de un tipo de datos especial que, además de ser el tipo por defecto, permite almacenar datos de cualquier otro tipo. La tabla 1 muestra los tipos de datos fundamentales de Visual Basic, así como su tamaño en bytes y el rango de valores admitido. Como alternativa a la utilización de la cláusula *As*, para indicar el tipo de una variable también es posible terminar su nombre con un determinado carácter a modo de sufijo.

Visual Basic dispone de tres tipos diferentes de módulos

```
Dim Contador As Integer, Nombre As String
Dim Contador%, Nombre$
```

Si en la sección de declaraciones del módulo se incluye la sentencia *Option Explicit,* se generará un error cada vez que se encuentre una variable no declarada.

NÚMEROS

Para representar números enteros se utilizan los tipos *Integer* y *Long,* según el tamaño. Para representar números reales se utilizan los tipos *Single* y *Double.* Además, existe un tipo numérico especial, denominado *Byte,* que permite representar octetos, es decir, números enteros sin signo en el rango 0 a 255.

```
Dim Edad As Integer, DNI As Long
```

TABLA 2

Función	Descripción
Abs	Devuelve el valor absoluto de un número
Atn	Devuelve el arcotangente de un número
Cos	Devuelve el coseno de un número
Exp	Devuelve el exponencial de un número en base e
Fix	Devuelve la parte entera de un número, redondeando los valores negativos al siguiente valor superior
Hex	Devuelve una cadena que representa el valor hexadecimal de un número
Int	Devuelve la parte entera de un número
Log	Devuelve el logaritmo natural de un número (en base e)
Oct	Devuelve una cadena de caracteres que representa el valor octal de un número
Rnd	Devuelve un número aleatorio menor que 1 y mayor o igual que 0
Sgn	Devuelve el signo de un número (-1 si es negativo, 0 si es nulo y 1 si es positivo)
Sin	Devuelve el seno de un número
Sqr	Devuelve la raíz cuadrada de un número
Tan	Devuelve la tangente de un número
Val	Devuelve el número representado por una cadena de caracteres

Funciones de tratamiento de datos numéricos.

CADENAS

El tipo de datos *String* (cadena) consiste en un conjunto de letras, números y símbolos especiales (códigos ASCII de 0 a 255) alineados unos detrás de otros. Visual Basic soporta dos tipos de cadenas diferentes:

- De tamaño variable: El tamaño de la cadena crece o decrece según se le van añadiendo o sustrayendo caracteres.
- De tamaño fijo: El tamaño de la cadena permanece constante desde su creación. Si se asigna una cadena de menor tamaño, se completa con espacios en blanco, mientras que si se asigna una de tamaño mayor, se trunca.

Al declarar cadenas de tamaño fijo, además de la palabra reservada *String*, debe especificarse el tamaño, utilizando para ello la sintaxis *As String * Tamaño.*

*Dim Nombre As String * 25*

Una de las principales operaciones que pueden realizarse con cadenas es la concatenación (operador &), que consiste en la generación de una única cadena resultado, a partir de la unión de otras cadenas operando.

Visual Basic dispone de una serie de funciones intrínsecas que permiten tratar los distintos tipos de datos

Nombre = "Daniel"
Print "Mi nombre es " & Nombre
'Resultado: Mi nombre es Daniel

También pueden realizarse operaciones de comparación entre cadenas mediante operadores relacionales. En este caso, la comparación se realiza carácter a carácter, atendiendo a su código ASCII. También es posible realizar comparaciones ignorando las mayúsculas y minúsculas, así como los signos de acentuación de las vocales. Para ello debe incluirse la sentencia *Option Compare Text* en la sección de declaraciones del módulo.

TABLA 5

Función	Descripción
Date	Devuelve la fecha actual del sistema
DateAdd	Devuelve una fecha añadiéndole un cierto intervalo de tiempo especificado
DateDiff	Devuelve el número de intervalos de tiempo entre dos fechas especificadas
DatePart	Devuelve un parte de una fecha especificada (el año, el día, los minutos, etc.)
DateSerial	Devuelve la fecha correspondiente a un determinado día, mes y año
DateValue	Convierte un expresión a una fecha válida
Day	Devuelve un número entre 1 y 31 representando el día del mes
Hour	Devuelve un número entre 0 y 23 representando la hora del día
Minute	Devuelve un número entre 0 y 59 representando los minutos de la hora
Month	Devuelve un número entre 1 y 12 representando el mes del año
Now	Devuelve la fecha y hora actual del sistema
Second	Devuelve un número entre 0 y 59 representando los segundos de minuto
Time	Devuelve la hora actual del sistema
Timer	Devuelve el número de segundos transcurridos desde la medianoche
TimeSerial	Devuelve la hora correspondiente a una determinado hora, minuto y segundo
TimeValue	Convierte un expresión a una hora válida
WeekDay	Devuelve un número que representa el día de la semana
Year	Devuelve un número entero representando el año

Funciones de tratamiento de fechas y horas.

El modo por defecto es *Option Compare Binary.*

Option Compare Text
"aaa" = "AAA"   'Resultado: True
"áàá" = "AAA"   'Resultado: True

También es posible realizar comparaciones de cadenas con un patrón. Para ello se utiliza el operador *Like*, que además de patrones de caracteres, admite los caracteres comodín del sistema operativo, así como otras múltiples funcionalidades que pueden consultarse en la ayuda *On-Line* de Visual Basic.
"Hola" Like "H" 'Resultado: True*

Visual Basic también dispone de una serie de funciones intrínsecas que permiten tratar cadenas de caracteres. La tabla 4 muestra una relación de dichas funciones.

Una variable de tipo Variant puede contener datos de cualquiera de los tipos soportados por Visual Basic

Print Chr(65)
'Resultado: A
Print "$" & Str(10 + 5)
'Resultado: $15
Print UCase("Unidad")
'Resultado: UNIDAD
Print Len("Mensaje")
'Resultado: 7
Print Mid("Mensaje", 3, 2)'Resultado: ns
Print String(5, "A")
'Resultado: AAAAA

FECHAS Y HORAS

Visual Basic dispone de un tipo de datos especial, denominado *Date*, que permite representar fechas y horas. Cada valor representa un día del calendario y una hora del día. Los valores admitidos para las fechas van del 1 de Enero del 100 al 31 de Diciembre del 9999, mientras que para las horas van de las 0:00:00 a las 23:59:59.

Los valores literales de fechas y horas deben expresarse en un formato reconocible por Windows, encerrados por el carácter sostenido (#). Los formatos reconocibles se especifican en el panel de control de Windows.
#15/05/95 14:30:15#
#15 Mayo 1995 14:30#

Al igual que para el resto de los tipos de datos, Visual Basic dispone de un conjunto de funciones para el tratamiento de los datos de tipo fecha. Estas funciones se muestran en la tabla 5.



Dim Estatura As Single, Distancia As Double
Dim Octeto As Byte

Los datos de tipo numérico pueden enlazarse entre sí mediante operadores aritméticos, generando así un nuevo dato numérico. De esta forma, es posible utilizar los operadores + (suma), - (resta), * (multiplicación), / (división), \ (división entera), *Mod* (módulo) y ^ (exponenciación), así como los paréntesis para alterar el orden de precedencia de las operaciones.

Num = (10 + Cont) Mod 3

También es posible enlazar datos numéricos mediante operadores relacionales. De esta forma, es posible utilizar los operadores <, <=, >, >=, = y <> (distinto).

Las instrucciones se forman mediante la combinación de palabras reservadas de Visual Basic con expresiones del lenguaje

If Form1.Left >= 10 Then
Text1.Top = 0
End If

Visual Basic también dispone de una serie de funciones intrínsecas que permiten tratar los datos de tipo numérico. La tabla 2 muestra una relación de dichas funciones.

Print Abs(-5) 'Resultado: 5
Print Sgn(-10) 'Resultado: -1
Print Int(-2.5) 'Resultado: -3
Print Fix(-2.5) 'Resultado: -2
Print Exp(1) 'Resultado: 0,271828182845905
Print Sqr(144) 'Resultado: 12

Para representar valores numéricos de tipo monetario puede utilizarse un tipo de datos especial, denominado *Currency* (Moneda), que permite expresar magnitudes monetarias con 4 decimales de exactitud. Los valores de este tipo se almacenan internamente como números enteros de 8 bytes, pero multiplicados por 10.000. De esta forma, se obtiene un número con 15 dígitos a la izquierda del punto decimal y 4 a la derecha.

DATOS LÓGICOS

Para representar valores lógicos se utiliza el tipo *Boolean*, que sólo puede tomar dos valores posibles, Verdadero y Falso, representados por las palabras reservadas *True* y *False*, respectivamente. Las expresiones de comparación entre operandos de cualquier tipo producen un resultado de Verdadero o Falso, por lo que pueden ser consideradas datos de tipo lógico o Boolean.

Dim Final As Boolean
Final = False
Final = Cont > 10

Los datos numéricos pueden ser convertidos directamente a datos lógicos. Para ello, el valor 0 se convierte a *False*, mientras que cualquier otro valor se convierte en *True*. Recíprocamente, los valores lógicos pueden ser convertidos en valores numéricos. En este caso, un valor *False* se convierte en 0, mientras que un valor *True* se convierte en -1.

Los datos de tipo lógico pueden enlazarse entre sí mediante operadores lógicos, generando así un nuevo dato de tipo lógico. Pueden utilizarse los operadores *And*, *Or*, *Not*, *Xor* (*Or* exclusivo), *Imp* (implicación) y *Eqv* (equivalencia). Las posibles combinaciones entre los valores de los operandos de entrada y el resultado de salida son muy limitadas, y se representan en lo que se denomina Tabla de Verdad del operador. La tabla 3 muestra las posibles combinaciones para cada operador lógico.

5 < 8 And Not 4 > 2 'Resultado: False
6 = 5 Or 6 <> 5 'Resultado: True
4 >= 3 Xor 2 <= 5 'Resultado: False

TABLA 3

Operando 1	Operando 2	And	Eqv	Imp	Or	Xor
True	True	True	True	True	True	False
True	False	False	False	False	True	True
False	True	False	False	True	True	True
False	False	False	True	True	False	False

Tablas de verdad de los operadores lógicos.

TABLA 4

Función	Descripción
Acs	Devuelve el código ASCII del primer carácter de una cadena
Chr	Devuelve el caráter asociado al código ASCII de un número
InStr	Devuelve la posición de la primera ocurrencia de una cadena dentro de otra
LCase	Devuelve una cadena con sus caracteres convertidos a minúsculas
Left	Devuelve un número especificado de caracteres desde la izquierda de una cadena
Len	Devuelve el número de caracteres de una cadena, o el espacio en bytes ocupado por una variable
LTrim	Devuelve una cadena, eliminando los espacios en blanco a la izquierda
Mid	Devuelve la subcadena de tamaño y posición especifcada dentro de una cadena
Right	Devuelve un número especificado de caracteres desde la derecha de una cadena
RTrim	Devuelve una cadena, eliminando los espacios en blanco a la derecha
Space	Devuelve una cadena formada por un número especificado de espacios en blanco
Str	Devuelve la representación de un número en forma de cadena
StrComp	Devuelve un valor que indica el resultado de una comparación entre cadenas
StrConv	Devuelve una cadena con diversos tipos de conversiones realizadas sobre ella
String	Devuelve una cadena compuesta de un carácter repetido un determinado número de veces
Trim	Devuelve una cadena, eliminando los espacios en blanco a izquierda y derecha
UCase	Devuelve una cadena con sus caracteres convertidos a mayúsculas

Funciones de tratamiento de cadenas de caracteres.

etc.).

Dim V
V = 15 *'Contiene un número: 15*
V = V & " veces" 'Contiene la cadena: "15 veces"

Para conocer el tipo de datos almacenado en una variable *Variant*, se utilizan las funciones *VarType* y *TypeName*. La primera devuelve un valor numérico que representa el tipo de datos, de acuerdo con los distintos tipos de contenido de la variable argumento. La segunda devuelve una cadena que especifica literalmente el nombre del tipo. La tabla 6 muestra los valores devueltos por ambas.

Dim Valor
Valor = 2 / 3
Print TypeName(Valor) 'Resultado: Double
Valor = #12/05/96#
Print TypeName(Valor) 'Resultado: Date

Cuando una variable de tipo *Variant* ha sido declarada pero no inicializada, contiene un valor especial denominado *Empty*. Aunque este valor no es equivalente al valor numérico 0, Visual Basic lo trata como 0 en un contexto numérico o como una cadena vacía ("") en un contexto de cadenas. Para averiguar si una variable ha sido inicializada puede utilizarse la función *IsEmpty*.

Dim Valor
Print IsEmpty(Valor) 'Resultado: True
Print TypeName(Valor) 'Resultado: Empty
Print Valor + 5 'Resultado: 5

Visual Basic dispone de otro valor especial para indicar que una variable no contiene ningún dato válido. Se trata del valor *Null*, que posee una serie de características particulares. Un resultado de este tipo puede obtenerse de dos formas:

- Mediante la asignación directa del valor *Null* a una variable.
- Como resultado de evaluar una expresión que contenga algún valor *Null*.

Para averiguar si una expresión contiene o no datos válidos puede utilizarse la función *IsNull*.

Dim Num1, Num2
Num1 = 10
Num2 = Null
Print Num1 + Num2
'Resultado: Null

En muchas ocasiones las aplicaciones necesitan utilizar datos de un tipo específico, por lo que será necesario realizar conversiones entre los distintos formatos disponibles. Para ello, Visual Basic dispone de un cierto repertorio de funciones que se muestran en la tabla 7.

El valor especial Null indica que una variable no contiene ningún dato válido

TABLA 7

Función	Descripción
CBool	Convierte una expresión al tipo Boolean
CByte	Convierte una expresión al tipo Byte
CCur	Convierte una expresión al tipo Currency
CDate	Convierte una expresión al tipo Date
CDbl	Convierte una expresión al tipo Double
CInt	Convierte una expresión al tipo Integer
CLng	Convierte una expresión al tipo Long
CSng	Convierte una expresión al tipo Single
CStr	Convierte una expresión al tipo String
CVar	Convierte una expresión al tipo Variant

Funciones de conversión de tipos de datos.

Dim Cantidad As Integer
Exacto = CDbl(Cantidad)
Valor = CVar(Cantidad)
Gasto = CCur(Cantidad)

Sin embargo, antes de realizar una conversión es conveniente verificar que el dato que se intenta convertir se corresponde con un formato admisible para dicha conversión. Para ello, Visual Basic dispone de las funciones *IsDate* e *IsNumeric*, que indican si una expresión puede ser convertida a un dato de tipo Fecha/Hora o Numérico, respectivamente.

Dim Valor As Variant, Fecha As Date
Valor = "12 Mayo 1996"
If IsDate(Valor) Then
Fecha = CDate(Valor)
Print Fecha 'Resultado: 12/05/96
End If

Igualmente, Visual Basic dispone de las funciones *IsArray*, *IsObject* e *IsError*, que permiten determinar si una variable contiene una matriz, un objeto o un valor de error, respectivamente.

CONCLUSIÓN

Con este artículo se inicia el análisis de los distintos elementos que componen el código de las aplicaciones desarrolladas con Visual Basic. Los conocimientos previos que el programador posea, tanto sobre versiones anteriores de Visual Basic como del propio lenguaje de programación Basic, pueden resultarle de gran utilidad. Sin embargo, podrá comprobar que son muchas las mejoras incorporadas en la versión 4.0 con el fin de facilitar la tarea de la codificación. Los tipos de datos, así como sus operadores y funciones de tratamiento, constituyen uno de los pilares básicos de cualquier lenguaje de programación. La explicación detallada de algunas de estas funciones de mayor complejidad puede obtenerse en la ayuda *On-Line* de Visual Basic. En los próximos artículos se analizarán el resto de los aspectos relacionados con la programación en Visual Basic. Además de analizar las variables, constantes y matrices, así como el control del flujo de programa, se hará especial hincapié en la creación y el manejo de objetos, ya que éste es uno de los aspectos que mas se ha potenciado con la última versión, acercando así Visual Basic a las más modernas técnicas de programación orientada a objetos.

TABLA 6

VaryType	TypeName	Contenido de la variable
0	Empty	Vacía (sin inicializar)
1	Null	Valor nulo (no válido)
2	Integer	Número entero de 2 bytes
3	Long	Número entero de 4 bytes
4	Single	Número en punto flotante de precisión simple
5	Double	Número en punto flotante de doble precisión
6	Currency	Magnitud monetaria
7	Date	Fecha/hora
8	String	Cadena de caracteres
9		Objeto de automatización OLE
10		Error
11	Boolean	Valor lógico
12		Variant
13		Objeto de tipo no OLE
17	Byte	Número entero de 1 byte
8192		Matriz

Valores devueltos por las funciones VarType y TypeName.

Print Date
'Resultado: 12/05/96
Print Month(Date)
'Resultado: 5
Print Time
'Resultado: 4:34:37
Print Hour(Time)
'Resultado: 4

El tipo de datos Date permite representar fechas y horas

Print DateAdd("m", 2, Date)
'Resultado: 12/07/96
Print DateSerial(1995, 15, 35)'Resultado: 4/04/1996

TIPOS DEFINIDOS POR EL USUARIO

Visual Basic permite al programador definir sus propios tipos de datos, mediante la combinación de los diferentes tipos de datos fundamentales que ya han sido estudiados. De esta forma, es posible declarar variables sencillas que contengan varios datos con distinto significado, e incluso de distinta naturaleza. Para ello, se utiliza la sentencia *Type*, cuya definición debe realizarse en la sección de declaraciones del módulo donde van a ser utilizados.

Type NombreTipo

NombreElemento As Tipo
NombreElemento As Tipo
. . .
End Type

Los elementos que componen la definición del tipo están constituidos por un nombre de variable, seguido por la especificación de uno de los tipos fundamentales de Visual Basic. Es posible incorporar elementos de cualquier tipo de datos, incluidos otros tipos de datos definidos por el usuario y matrices. En este último caso, debe especificarse el rango y las dimensiones.

Type Persona
Nombre As String
DNI As Long
FechaNac As Date
End Type

Una vez realizada la definición, es posible declarar variables del tipo de datos Persona, usando para ello la sentencia *Dim* correspondiente. El acceso a los valores individuales de los elementos se realiza especificando el nombre de la variable (no del tipo) y el del elemento a acceder, separados por un punto, de forma similar a como se accede a las propiedades de un objeto. También es posible asignar una variable de un tipo definido por el usuario a otra, siempre y cuando ambas procedan del mismo tipo.

En este caso, el valor de cada elemento de una variable se asigna al correspondiente de la otra.

Dim Yo As Persona, Tu As Persona

Yo.Nombre = "JUAN"

Yo.DNI = 12345678
Tu.FechaNac = #15/05/64#

Yo = Tu

EL TIPO VARIANT

Como ya se mencionó anteriormente, Visual Basic dispone de un tipo de datos que permite almacenar datos de cualquiera de los otros tipos, denominado *Variant*. Se trata de un tipo de datos que no es necesario declarar explícitamente. Una variable de este tipo puede contener datos de cualquiera de los tipos soportados por Visual Basic, salvo cadenas de tamaño fijo y tipos definidos por el usuario.

Los operadores y funciones aplicables a datos de este tipo son todos los aplicables a los tipos de datos que puede contener (operaciones aritméticas, concatenaciones, operaciones lógicas,

menús, aceleradores de teclas y barras de herramientas. *CTreeCtrl* suele utilizarse en cajas de diálogo con otros controles, donde todas las ventajas anteriores dejan de tener importancia.

USANDO LOS CONTROLES *TREE VIEW*

El método más habitual para la utilización de un control *TreeView* con Visual C++ 4.0 es el siguiente:

El control debe de ser creado. En el caso de estar utilizando el control en una caja de diálogo o una vista, el control será creado de forma automática, pero si se quiere crear el control como ventana hija de otra, se debe llamar a la función miembro *Create* de la clase *CTreeCtrl*.

En el siguiente paso, y si se desea que los nodos del árbol dispongan de una imagen o icono, hay que asignar al árbol una lista de imágenes, que permitan mostrar las diferentes imágenes en los diferentes nodos del árbol. Además, se puede cambiar el tabulado entre elementos padres e hijos con la función *SetIndent*. Estos pasos se suelen realizar en las funciones miembro *OnInitDialog*, en el caso de cajas de diálogo, y en *OnInitialUpdate* en caso de Vistas.

La siguiente acción a tomar es insertar los elementos del árbol. Para ello se llama a la función *InsertItem*. Esta función devuelve el *handle* al elemento insertado, para poder localizarlo a posteriori, en el caso de que se le quieran por ejemplo añadir nuevos elementos, o simplemente se le desee cambiar la etiqueta. Al igual que en caso anterior, las funciones *OnInitDialog* o *OnInitialUpdate* son el mejor sitio para incluir estos pasos.

Una vez creado el control y cuando la aplicación este ejecutándose, el usuario realizará operaciones sobre el árbol, lo que implica que se envíen mensajes de notificación a la aplicación. A cada acción o mensaje de notificación que se quiera atender se debe añadir una función miembro, y la forma más sencilla es desde *ClassWizard* o *ClassBar*, que crean automáticamente el código necesario para incluir la macro *ON_NOTIFY_REFLECT* y la estructura de la función a utilizar.

Para actuar sobre el control se debe llamar a las distintas funciones miembro de la clase, que permiten cambiar la etiqueta, la imagen o los datos asociados con cada uno de los elementos del árbol. Del mismo modo, existen otra cantidad de funciones que permiten obtener información relacionada con cada elemento o con el árbol en general.

Una vez que la utilización del control ha finalizado hay que destruirlo. Para ello, y si el control está en un diálogo o en una vista, no hace falta ninguna operación, ya que del mismo modo que en la creación, el control y su clase *CTreeCtrl* se destruyen automáticamente. En caso contrario, habrá que asegurarse de destruir el control y la clase que lo gestiona.

COMUNICACIÓN CON EL *TREE VIEW CONTROL*

Para comunicarse con el control se debe llamar a las numerosas funciones miembro que tiene la clase *CTreeCtrl*. Por tanto, para establecer dicha comunicación se debe disponer de la dirección del objeto de la clase *CTreeCtrl* que gestiona al control. En función del modo de utilización del control hay tres modos diferentes de obtener la dirección del objeto.

- En caso de que el árbol sea una ventana hija de una caja de diálogo, existirá una variable miembro del tipo *CTreeCtrl* en el objeto que gestiona al diálogo, que gestione a su vez las acciones sobre el control del árbol.
- En el caso de que el árbol sea una ventana hija, la dirección se obtiene

Para comunicarse con el control hay que llamar a las numerosas funciones miembro que tiene la clase CTreeCtrL

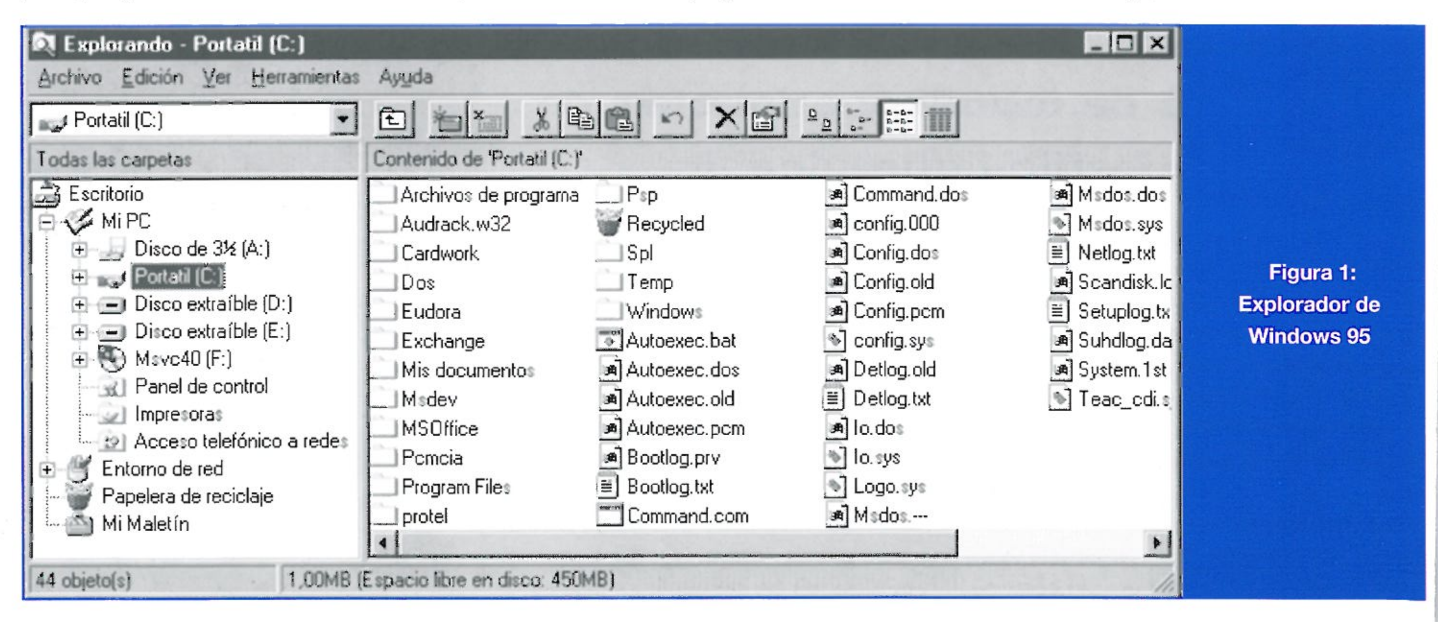
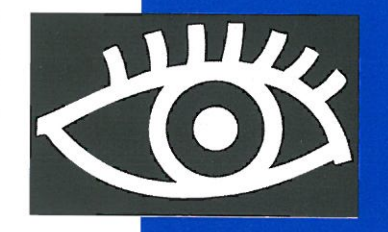


Figura 1: Explorador de Windows 95

EL CONTROL DE WINDOWS TREECTRL

Jorge R. Regidor



En Windows 95 se incluyen una serie de nuevos controles aumentando los ya existentes. Estos controles amplían de forma notable la funcionalidad que brindaban los ya existentes en Windows 3.x como las listas, las *comboboxes,* los botones, el editor o las barras de *scroll*.

Uno de estos nuevos controles permite mostrar una lista de iconos y etiquetas de forma jerárquica, al modo que la parte izquierda del explorador de carpetas de Windows 95 (Ver Figura 1). Este control se denomina *Tree View Control*, y Visual C++ lo encapsula bajo la clase *CTreeCtrl*.

Esta lista jerárquica de elementos con etiquetas y opcionalmente iconos o bitmaps puede tener subelementos, los cuales son mostrados u ocultados al hacer doble click sobre el elemento en cuestión. Estos controles permiten, por ejemplo, mostrar la arquitectura del árbol de directorios de una unidad de disco o el índice de un libro electrónico (Ver Figura 2). Como cada uno presupone, este control tiene muchos usos en muy diferentes campos, pero siempre o casi siempre relacionados con la visión de estructuras jerárquicas.

Este control, junto a todos los nuevos controles de Windows 95, se encuentran en una DLL llamada COMMCTRL.DLL, y cada uno de ellos tiene un interfaz de funciones, una estructura y unos mensajes para su manipulación. Visual C++, mediante sus clases de la librería MFC, encapsula cada uno de estos controles para permitir una programación más sencilla e intuitiva.

LA CLASE *CTREECTRL*

La clase *CTreeCtrl* es la encargada de ofrecer una forma más sencilla, intuitiva y orientada a objetos, de utilización del control *Tree View*. Para ello esta clase encapsula todas la operaciones sobre el control bajo las funciones miembro, almacena los datos de estado sobre sus variables miembro, y facilita su utilización al disponer en *ClassBar* y *ClassWizard* de la lista de todos los mensajes que puede recibir dicho control.

LA CLASE CCTRLVIEW

Además de la clase *CTreeCtrl*, existe otra llamada *CTreeView* que también permite utilizar el control *Tree View* de Windows. Esta clase actúa como una vista dentro de la arquitectura documento/vista, lo que permite mostrar documentos con un tipo de estructura jerárquica.

La utilización de *CTreeView* frente a *CTreeCtrl* pasa por la necesidad de que el *Tree View* necesite utilizar las propiedades típicas de una vista, como ocupar automáticamente el área de cliente de la ventana de marco (*Frame Window*), la necesidad de procesar mensajes de

El control Tree View permite mostrar una lista de iconos y etiquetas de forma jerárquica

Uno de los puntos fuertes que Visual C++ 4.0 y las MFC ofrecen es la encapsulación bajo clases de los controles estándar que Windows proporciona, tanto los anteriores de Windows 3.x como los nuevos de Windows 95.

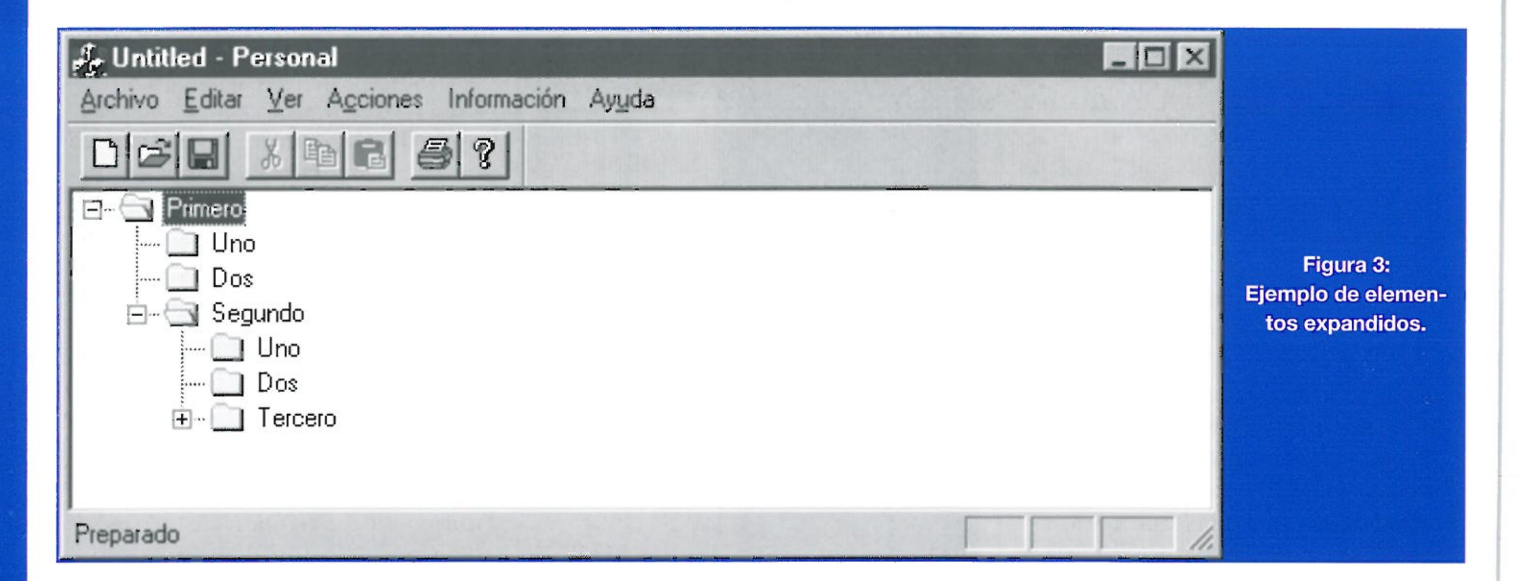


Figura 3: Ejemplo de elementos expandidos.

lugar. Por último, el valor *TVI_SORT* permite insertar el elemento ordenado alfabéticamente.

EDITANDO LAS ETIQUETAS DE LOS ELEMENTOS

En un control *Tree View* con el estilo *TVS_EDITLABELS,* un usuario puede editar la etiqueta del elemento que está seleccionado. Para ello sólo tiene que hacer click en la etiqueta. Además, desde la aplicación se puede hacer lo mismo llamando a la función miembro *EditLabel.*

Antes de que se comience a editar una etiqueta de un elemento se recibe el mensaje de notificación *TVN_BEGINLABELEDIT*, que permite aceptar o no la edición de las etiquetas de un elemento. Además, en este momento es cuando se crea el objeto que controla el proceso de edición (la clase *CEdit*), lo que permite actuar sobre dicho objeto y, por ejemplo, limitar la cantidad de caracteres a introducir.

Una vez ha finalizado la edición de la etiqueta, se recibe el mensaje de notificación *TVN_ENDLABELEDIT.* Se debe atender a este mensaje si se quiere actualizar la etiqueta del elemento, ya que es responsabilidad del programador validar la cadena introducida, así como sustituirla en el elemento.

ESTADOS DE LOS ELEMENTOS

Cada uno de los elementos de un árbol tiene asignado un estado. Algunos estados se actualizan automáticamente a medida que el usuario actúa sobre el control y sus elementos, como por ejemplo, la selección o no de un elemento. Pero además existe una función para modificar el estado de un elemento y otra para saber el estado actual. La tabla 2 muestra una lista de los diferentes estados.

Cuando se llama a la función para modificar el estado de un elemento, hay que especificar en el campo *nStateMask* los bits que se quieren cambiar y en *nState* el nuevo valor. Por ejemplo, si se quiere cambiar la imagen de estado, se indica en *nStateMask* el valor *TVIS_STATEIMAGEMASK*, y en el campo *nState* se indica el índice del la lista de imágenes a activar.

LISTAS DE IMAGENES DEL CONTROL *TREE VIEW*

Cada elemento de un control *Tree View* pude tener un par de imágenes asociadas a él. Una imagen es mostrada cuando el elemento está seleccionado, y la otra cuando el elemento no está seleccionado.

Para utilizar imágenes dentro de un control *Tree View* hay que crear un objeto del tipo *CImageList*, añadir las diferentes imágenes y asociarlo al control *Tree View*. A cada elemento se le puede asignar un par de imágenes de la lista.

También se puede crear un lista de imágenes de superposición, que será superpuesta sobre las imágenes en cada elemento. Para seleccionar cada imagen de superposición hay que indicar un nuevo estado con la máscara *TVIS_OVERLAYMASK*, y en el estado indicar el índice de la imagen dentro de la lista de imágenes de superposición.

Por último, se puede crear una lista de imágenes de estado, que permiten mostrar a la izquierda de la imagen de cada elemento una imagen de estado. Se puede utilizar, por ejemplo, para mostrar el estado de cada elemento con *check boxes.*

SELECCIÓN DE ELEMENTOS

Cuando se cambia la selección de un elemento a otro, se reciben los mensajes de notificación *TVN_SELCHANGING* y *TVN_SELCHANGED.* Los mensajes incluyen información de si el cambio es fruto de una operación de teclado o del ratón, e incluyen también información de los dos elementos implicados, el que pierde la selección y el que la gana. De este modo se pueden actualizar los estados de cada uno de ellos. Si se quiere cambiar la selección en la aplicación, se puede hacer llamando a la función *SelectItem.*

INFORMACIÓN SOBRE LOS ELEMENTOS

La clase *CTreeCtrl* tiene una serie de funciones que permiten recoger infor-

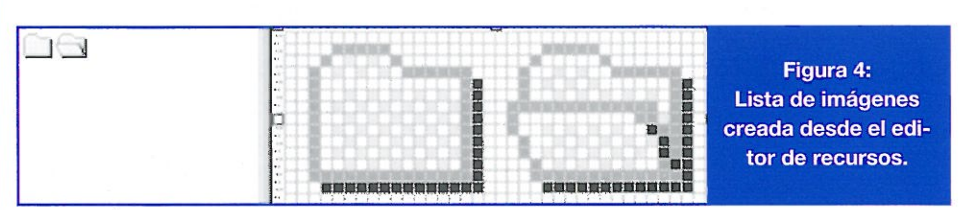


Figura 4: Lista de imágenes creada desde el editor de recursos.

al crear el objeto a dicha clase, por lo que dicho puntero debe ser accesible en cualquier sitio que se deba utilizar.

- El último caso, y quizá el único un tanto diferente, se presenta al utilizar el control árbol en una vista, es decir, se utiliza la clase *CTreeView*. Esta clase dispone de una función miembro, que permite acceder al objeto *CTreeCtrl* de la clase *CTreeView*. Esta función es *CTreeView::GetTreeCtrl*, que de hecho es la única función exclusiva de esta clase frente a su clase madre *CCtrlView*.

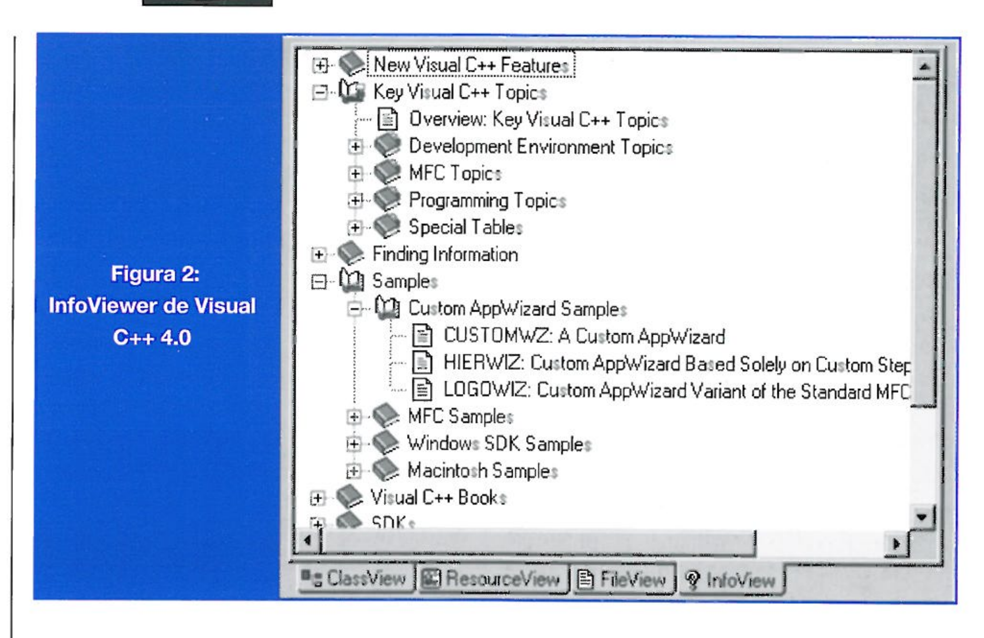


Figura 2: InfoViewer de Visual C++ 4.0

ESTILOS DEL CONTROL *TREE VIEW*

Como cualquier otra ventana en Windows, el control *Tree View* dispone de una serie de estilos que modifican el funcionamiento y el aspecto del control (Ver Tabla 1). También, como en cualquier otra ventana, se pueden obtener y cambiar estos estilos con las funciones *GetWindowLong* y *SetWindowLong* respectivamente. En el control *Tree View*, existen cuatro estilos que son *TVS_HASLINES*, *TVS_HASBUTTONS*, *TVS_LINESATROOT* y *TVS_EDITLABELS*.

El estilo *TVS_HASLINES* permite mostrar una línea que enlaza a un elemento padre con sus elementos hijos. Este estilo no enlaza elementos en el primer nivel de jerarquía. Para permitir líneas en este primer nivel se debe definir el estilo *TVS_LINESATROOT*. Otro estilo es *TVS_HASBUTTONS*, que muestra unos botones en la parte izquierda de cada elemento que tenga a su vez elementos hijos. Para visualizar o no los elementos hijos se debe hacer un doble click sobre dicho elemento, pero si dispone de un botón, basta con hacer un click sobre él. Del mismo modo que en el caso anterior, *TVS_HASBUTTONS* no muestra botones en el primer nivel de la jerarquía, a no ser que se combine con los estilos *TVS_HASLINES* y *TVS_LINESATROOT*. El último estilo es *TVS_EDITLABELS*, que permite al usuario editar las etiquetas de los elementos.

El control Tree View dispone de una serie de estilos que modifican el funcionamiento y el aspecto del control

ELEMENTOS E HIJOS DE UN CONTROL *TREE VIEW*

Cualquier elemento de un árbol puede a su vez tener una lista de elementos hijos asociados a él. Estos elementos que tienen uno o más hijos se denominan elementos padre. Además, los elementos que no tienen padres se llaman elementos raíz. Para indicar que un elemento es hijo de otro se muestra el hijo desplazado hacia la derecha unos bytes sobre el elemento padre.

En todo momento, el estado de la lista de elementos hijos de un elemento padre puede estar expandida o colapsada. En modo expandido se muestran todos los elementos hijos, y en modo colapsado no se muestran.

Cuando se añade un elemento al árbol, se debe indicar en que posición se quiere añadir, o lo que es lo mismo, cuál es el elemento padre del que se desea añadir. En el caso de que se quiera añadir un elemento raiz, se deberá indicar pasándole como elemento padre NULL o *TVI_ROOT*.

Como ya se ha indicado anteriormente, cuando se hace un doble click sobre un elemento padre, los elementos hijos invierten su estado entre colapsado y expandido dependiendo del estado anterior. Antes de que esto ocurra se recibe un mensaje de notificación, *TVN_ITEMEXPANDING*, por si se desea impedir o bien tomar alguna decisión. Después de que la operación de expandir o colapsar se haya realizado, se recibe el mensaje de notificación *TVN_ITEMEXPANDED* indicando tal acontecimiento.

En la figura 3 se muestra un árbol con los elementos Primero y Segundo expandidos y el elemento Tercero colapsado.

POSICIÓN DE LOS ELEMENTOS

Cuando se añade un elemento, ya se ha indicado cómo hacerlo en una determinada posición de la jerarquía, pero también puede hacerse en una determinada posición dentro de las lista de elementos hijos. Para ello, dentro de la función *InsertItem* existe un segundo *handle* donde se le puede indicar el elemento después del que se va a insertar el nuevo. Existen tres valores especiales para este parámetro, que son *TVI_FIRST*, *TVI_LAST* y *TVI_SORT*.

El valor *TVI_FISRT* permite insertar el nuevo elemento en primera posición de la lista de elementos hijos. El valor *TVI_LAST*, de modo análogo, permite insertar el nuevo elemento en último

ESTADO	DESCRIPCIÓN
TVIS_BOLD	El elemento está en negrita.
TVIS_CUT	El elemento es parte de una operación de cut y paste.
TVIS_DROPHILITED	El elemento es destino de una operación drag and drop.
TVIS_EXPANDED	El elemento esta expandido.
TVIS_EXPANDEDONCE	El elemento tiene un hijo a su vez expandido.
TVIS_FOCUSED	El elemento tiene el foco.
TVIS_OVERLAYMASK	Máscara para poner la imagen de superposición.
TVIS_SELECTED	El elemento está seleccionado.
TVIS_STATEIMAGEMASK	Máscara para seleccionar la imagen de estado.
TVIS_USERMASK	Igual que TVIS_STATEIMAGEMASK

Tabla 2: Estados del control Tree View

para atender mensajes de notificación como *TVN_ENDLABELEDIT*. El siguiente listado muestra cómo se atiende a dicho mensaje y se comprueba que la cadena no tiene longitud cero.

```
void CPersonalView::OnEndLabelEdit
(NMHDR* pNMHDR, LRESULT*
pResult)
{
TV_DISPINFO* pTVDispInfo =
(TV_DISPINFO*)pNMHDR;

if(lstrlen(pTVDispInfo->item.pszText)
> 0)
{
PersonalTree->SetItemText(pTVDispInfo-
>item.hItem,
pTVDispInfo->item.pszText);
}
*pResult = 0;
}
```

Para utilizar imágenes dentro de un control Tree View hay que crear un objeto del tipo CImageList

También se ha atendido al mensaje *TVN_BEGINLABELEDIT*, con el fin de limitar a 20 los caracteres de la etiqueta. Para ello, se tiene que indicar al objeto de edición que sólo acepte 20 caracteres.

```
void CPersonalView::OnBeginLabelEdit
(NMHDR* pNMHDR, LRESULT*
pResult)
{
TV_DISPINFO* pTVDispInfo = (TV_
DISPINFO*)pNMHDR;
PersonalTree->GetEditControl()-
>SetLimitText(20);
*pResult = 0;
}
```

Además, también se atiende al mensaje *TVN_ITEMEXPANDED*, para actualizar la imagen, de tal modo que cuando una carpeta esté expandida, la carpeta esté abierta.

```
void CPersonalView::OnItemExpanded
(NMHDR* pNMHDR, LRESULT*
pResult)
{
NM_TREEVIEW* pNMTreeView =
(NM_TREEVIEW*)pNMHDR;
if(pNMTreeView->action == TVE_
EXPAND)
PersonalTree->SetItemImage
(pNMTreeView->itemNew.hItem, 1, 1);
else
PersonalTree->SetItemImage
(pNMTreeView->itemNew.hItem, 0, 1);
*pResult = 0;
}
```

Además de estas funciones, se deben definir los estilos del control *Tree View*, y para ello se sobrecarga la función *PreCreateWindow* de la clase *CTreeView*.

```
BOOL
CPersonalView::PreCreateWindow
(CREATESTRUCT& cs)
{
cs.style |= (TVS_LINESATROOT |
TVS_HASLINES |
TVS_HASBUTTONS | TVS_EDITLA-
BELS);
return CTreeView::PreCreateWindow
(cs);
}
```

En esta función se modifica el campo *style* de la estructura del tipo *CREATESTRUCT* utilizada para crear el control *CTreeCtrl*. Como se ve en el listado anterior, se han seleccionado los estilos *TVS_LINESATROOT*, *TVS_HASLINES*, *TVS_HASBUTTONS* y *TVS_EDITLABELS*.

CONCLUSIÓN

En este articulo se ha pretendido introducir el método de trabajo con la clase *CTreeCtrl*. El método es igual con este control que con los demás controles, la única diferencia radica en los propios mensajes y las funciones de la clase.

Como es lógico, el articulo sólo se introduce en los primeros pasos de programación del control, por lo que se emplaza a cada lector a que ahonde en el control y sus métodos, con el fin último de sacarle el máximo rendimiento a un potente control.

PRÓXIMA ENTREGA

El mes próximo se tratará otro tipo de control y para ello se verá desde una perspectiva distinta. Por ello, se va a crear un diálogo donde estará incluido y se tratará de explicar los métodos básicos para su programación con éxito.

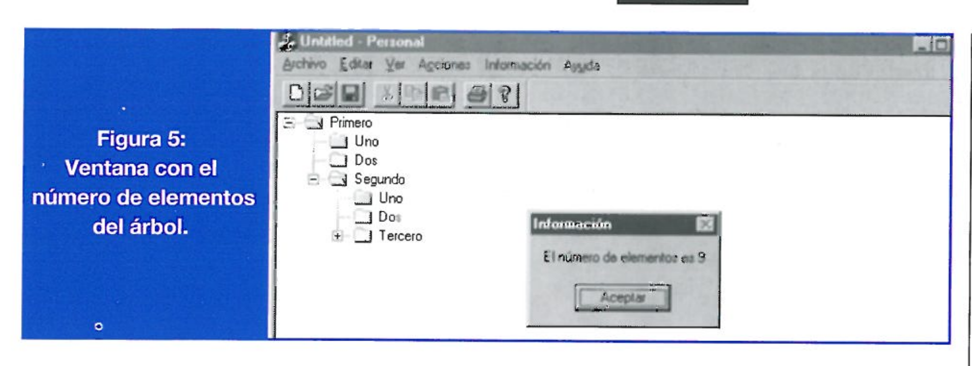


Figura 5: Ventana con el número de elementos del árbol.

mación de cada uno de los elementos que componen el árbol. La función *GetItem* devuelve toda la información asociada con un elemento.

Hay varias funciones que permiten obtener información sobre diferentes partes del elemento. Estas son *GetItemState*, que obtiene el estado del elemento; *GetItemText*, que obtiene la etiqueta del elemento; *GetItemData*, que obtiene los datos del elemento y *GetItemImage*, que obtiene la imagen asociada al elemento.

Además, hay diversas funciones para poder moverse por los diferentes elementos del árbol. Hay funciones como *GetNextItem* que devuelve el próximo elemento, *GetRootItem* que devuelve el elemento raíz, *GetNextVisibleItem*, *GetPrevVisibleItem*, *GetParentItem*,...

La función *GetItemRect* devuelve el rectángulo ocupado por el elemento indicado. *GetCount* devuelve el número de elementos del árbol y *GetVisibleCount* devuelve el número de elementos visibles en la ventana. Además, se puede averiguar si un elemento es visible con la función *EnsureVisible*.

MENSAJES DE NOTIFICACIÓN

El control *Tree View* envía una serie de mensajes de notificación que permiten a la aplicación conocer en todo momento las acciones del usuario. Los mensajes de notificación definidos para este control se muestran en la tabla 3.

- El mensaje *TVN_BEGINDRAG* indica el inicio de una operación *drag&drop*.
- El mensaje *TVN_BEGINLABELEDIT* indica el inicio de la edición de la etiqueta de un elemento, así como *TVN_ENDLABELEDIT* indica el final de la edición.
- El mensaje *TVN_DELETEITEM* indica que un elemento es eliminado.
- El mensaje *TVN_GETDISPINFO* pregunta por la información necesaria por el para que el control *Tree View* pueda dibujar el elemento. Con *TVN_SETDISPINFO* se actualiza la información mantenida para un elemento.
- *TVN_ITEMEXPANDED* indica que el padre de un elemento ha sido expandido o colapsado.
- *TVN_KEYDOWN* indica que una tecla ha sido pulsada.
- *TVN_SELCHANGING* indica que se va a producir un cambio en la selección de un elemento a otro. El mensaje *TVN_SELCHANGED* indica que el cambio de selección ya se ha producido.

EL PROGRAMA DE EJEMPLO

Para ilustrar el método de programación del control *Tree View*, se ha creado una aplicación desde *AppWizard* con una vista *CTreeView*.

En primer lugar se deben insertar los elementos necesarios, y para ello hay que llamar a la función *InsertItem* dentro de la función *OnInitialUpdate*. Las siguientes líneas muestran este proceso:

PersonalTree = &GetTreeCtrl();
PersonalTree->SetImageList (PersonalImage, TVSIL_NORMAL);
Primero = PersonalTree->InsertItem ("Primero", 0, 1);

PersonalTree es una variable miembro del tipo puntero a *CTreeCtrl*, donde se almacena la dirección del control *Tree View* de la clase *CTreeView*. Para almacenar ésta dirección se llama a la función miembro de la clase *CTreeView*, *GetTreeCtrl*.

En un control Tree View con el estilo TVS_EDITLABELS un usuario puede editar la etiqueta del elemento que está seleccionado

Posteriormente, se carga la lista de imágenes creada con anterioridad, indicando el parámetro *TVISL_NORMAL*, que indica que la lista de imágenes será utilizada para mostrar entre seleccionado y no seleccionado.

Por último, se llama a la función *InsertItem* con el primer parámetro la etiqueta del elemento, el segundo el índice de la lista de imágenes para estado no seleccionado y el tercero el índice para estado seleccionado. La función devuelve el *handle* del elemento insertado, útil si se quiere añadir otros elementos hijos a éste.

Se han añadido funciones miembro

ESTILO	DESCRIPCIÓN
TVS_HASLINES	Muestra unas líneas que unen los elementos hijos.
TVS_HASBUTTONS	Muestra botones en los elementos con hijos.
TVS_LINESATROOT	Muestra líneas entre elementos del primer nivel.
TVS_EDITLABELS	Permite editar las etiquetas de los elementos.

Tabla 1: Estilos del control Tree View

seleccionar las tres bases de datos de ejemplo y, por último, *tutorial* se utiliza para instalar el archivo PEOPLE.TXT usado en el tutorial del programa.

Una vez seleccionados los componentes y pulsado el botón *Next* se procede a la copia de archivos al disco duro. Al finalizar la misma, si se optó por instalar el lector Acrobat Reader, se pide al usuario que confirme la instalación del mismo. Pulsando el botón *Finish* acaba la instalación de 4D y se ejecuta el programa de instalación de Acrobat Reader. En primer lugar se muestra la licencia del producto que, una vez leída, se puede aceptar con el botón *Accept* o declinar con el botón *Decline*. Pulsando el botón *Accept* aparece una ventana que permite cambiar el directorio destino por defecto. Pulsando el botón *Install* se pide al usuario que introduzca su nombre y el nombre de su organización y se procede a la instalación del programa y a la creación del grupo de programas correspondiente. Al finalizar se reinicia Windows para que los cambios producidos en la configuración surtan efecto.

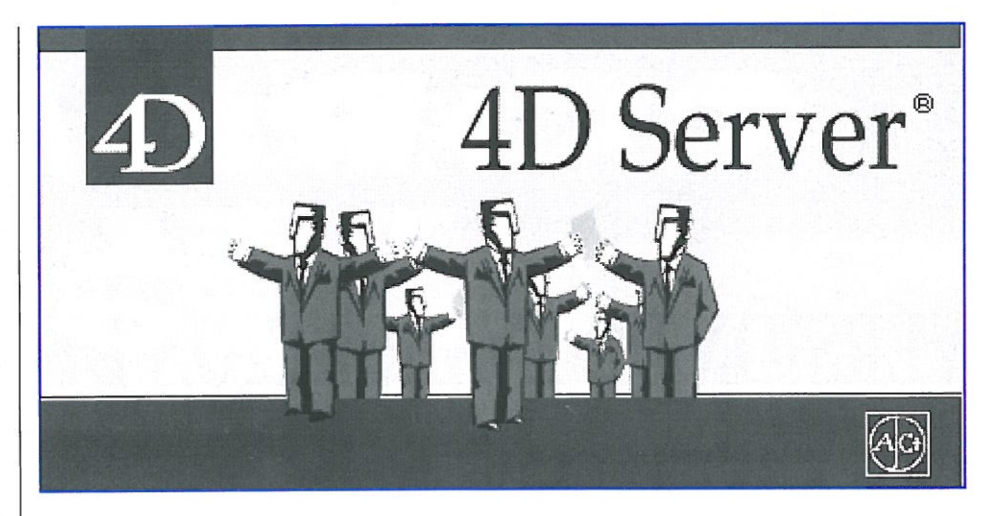
De todos modos, Acrobat Reader puede ser instalado en cualquier momento ejecutando el programa ACROREAD.EXE situado en el directorio \ACROBAT\ENGLISH\ACROWIN del CD-ROM.

Las bases de datos de ejemplo se instalan en el directorio EXAMPLES que cuelga del directorio de instalación del programa.

4D Server y 4D Client

Estos dos programas son los utilizados para implementar la arquitectura de base de datos cliente/servidor en red. 4D Server se ejecuta en el ordenador servidor y contiene las bases de datos a las que accederán los demás usuarios. 4D Client es el cliente y se instala en cada uno de los ordenadores desde los cuales se podrá acceder a las bases de datos. Los clientes realizan peticiones al servidor, que proporciona la información y los servicios necesarios. En el CD-ROM se incluye la versión 1.5.1 de 4D Client y 4D Server. Al igual que 4D, también se dispone de dos meses para evaluar el producto.

La instalación de ambos programas es análoga a la indicada en el apartado anterior. Para instalar 4D Server se debe ejecutar el programa SETUP.EXE del directorio \4DSERVER\ENGLISH\ DISK1. Los componentes de instalación son prácticamente iguales a los de 4D, con un fichero más de ayuda y sin el contenedor de utilidades OLE. Es obligatorio instalar el componente *4D Server Evaluation*, sin el cual no puede ejecutarse el programa. 4D Server necesita unos 13 megas libres en el disco duro para su instalación.



El cliente 4D Client se instala con el programa SETUP.EXE situado en el camino \4DCLIENT\ENGLISH\DISK1. En este caso, aparece un nuevo grupo de componentes denominado *Network* con los componentes de red ADSP, IPX/SPX y TCP/IP necesarios para la ejecución del cliente. No se debe olvidar la selección del componente 4D Client. El programa ocupa un total de 8 megas una vez instalado en el disco duro.

En los grupos de programas creados en la instalación de 4D Client y 4D Server existe un icono denominado "Network Component Doc". Este icono permite el acceso al archivo de Acrobat Reader que contiene la documentación de los componentes de red.

DOCUMENTACIÓN

Como ya se indicó anteriormente, para leer la documentación incluida en el CD-ROM se necesita el lector Acrobat Reader. Los archivos legibles por el mismo tienen la extensión PDF y están situados en los directorios \BROCHURE\ENGLISH, \DOC\ENGLISH Y \DOC\ENGLISH\INSTALL. A continuación se especifica el contenido de cada uno de estos archivos:

Directorio \BROCHURE\ENGLISH:
- Archivo MKG_INTL.PDF: es un folleto on-line con información sobre ACI y sus productos.

Directorio \DOC\ENGLISH:
- Archivo ADDENDUM.PDF: contiene información de última hora.
- Archivo DESIGN.PDF: consideraciones de diseño con el producto.
- Archivo LANGUAJE.PDF: describe el lenguaje utilizado en 4D.
- Archivo QSTART.PDF: guía de introducción rápida.

NOTA: Al leer este documento con Acrobat Reader, si se pincha en la ventana derecha sobre el nombre del capítulo que se desea leer, éste no será visualizado, excepto el capítulo 1 y el índice, que sí se visualizan. De todas formas, se puede acceder a cada capítulo a través de las opciones ubicadas en la ventana derecha, que sí permitirán realizar esta operación.

- Archivo SERVREF.PDF: información sobre el uso de 4D Server y 4D Client.
- Archivo TUTORIAL.PDF: tutorial del programa.
- Archivo USER.PDF: Referencia del entorno de usuario utilizado en 4D.

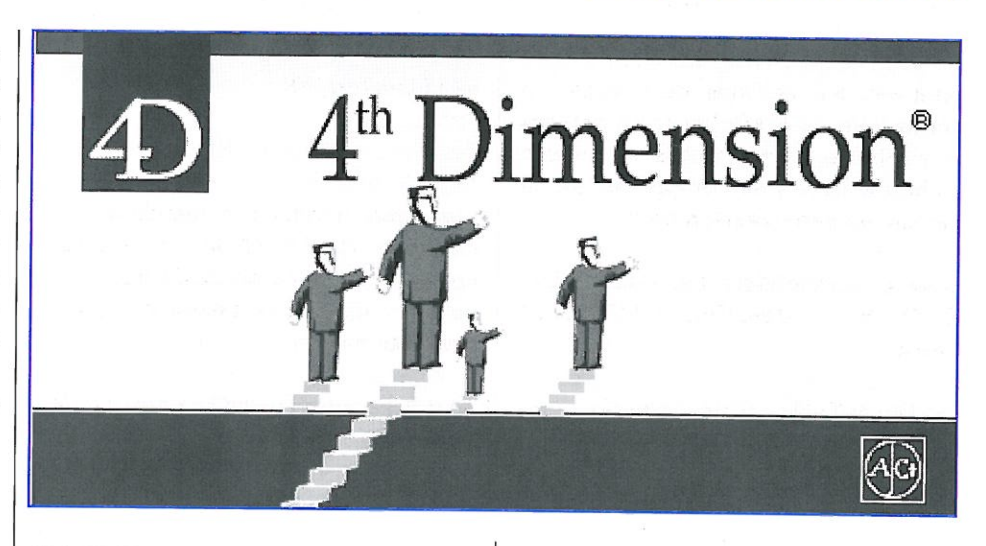
Directorio \DOC\ENGLISH\INSTALL:
- Archivo 4DINSTAL.PDF: guía de instalación de 4th Dimension.
- Archivo 4DSERINS.PDF: guía de instalación de 4D Server.

ARTÍCULOS

Como viene siendo habitual, en el CD-ROM que acompaña a la revista se incluyen los archivos correspondientes a los artículos publicados en el presente número. Dichos archivos se pueden encontrar, organizados en diferentes subdirectorios, en el directorio \DISK25 del disco.

VERSIÓN DE EVALUACIÓN DE 4TH DIMENSION

Junto con el presente número de Sólo Programadores se incluye un CD-ROM con las versiones de evaluación para Windows de 4D y 4D Server. Se trata de un sistema gestor de bases de datos relacionales cliente/servidor de 32 bits con un atractivo interfaz gráfico. Incorpora un potente lenguaje de cuarta generación y un compilador multiplataforma que permite mejorar el rendimiento de las aplicaciones y las bases de datos. Además, las aplicaciones creadas con 4D son independientes de la plataforma en que fueron creadas y pueden ejecutarse en las versiones del producto para otras máquinas.



REQUERIMIENTOS

Para instalar 4D se necesita un ordenador compatible PC equipado con un procesador 386 o superior y tarjeta de vídeo compatible con VGA. Los requerimientos de memoria dependen del entorno en el que se vaya a utilizar el producto. Con Windows 3.1 y Windows 95 basta con disponer de 8 megas de memoria, mientras que con Windows NT son imprescindibles 16 megas de RAM.

Además, 4D Server y su complemento, 4D Client, necesitan de una red para funcionar correctamente.

4th Dimension

4D es el producto principal de la familia de productos 4D. Permite la creación de aplicaciones en un sólo ordenador que más tarde pueden ser utilizadas en un entorno cliente/servidor mediante 4D Server y 4D Client. La versión incluida en el CD-ROM es la 3.5.1 y se dispone de dos meses para su evaluación.

La instalación del programa necesita de 15 megas libres en el disco duro y se realiza con la ejecución del programa SETUP.EXE situado en el directorio \4D\ENGLISH\DISK1.

Lo primero que aparece en pantalla es una ventana con la información referente a la licencia de la versión de evaluación. Para aceptar los términos de la licencia se debe pulsar el botón YES, con lo que continua la instalación.

En algunos equipos se han detectado problemas durante la ejecución de 4TH Dimension en Windows 3.1. El programa funciona correctamente aunque aparezca en pantalla un mensaje de error. Esto se debe a conflictos con otros controladores del sistema.

A continuación se pregunta al usuario por el tipo de instalación a realizar. El botón *Browse* sirve para cambiar el directorio de instalación del programa, por defecto C:\4DEV351. Existen dos opciones de instalación, la estándar (*Standard*) y la personalizada (*Custom*). La primera instala 4D directamente, mientras que la segunda permite seleccionar las partes del producto que serán instaladas. Una vez elegido el tipo de instalación se pulsa el botón *Next* para continuar. El botón *Cancel* sirve para cancelar la instalación y el botón *Back* (ahora sombreado) se utiliza para volver atrás en cualquier momento.

Si se ha elegido la instalación personalizada la siguiente ventana muestra una lista con los componentes principales de 4D y el espacio que ocupan en disco. Pinchando con el ratón sobre cualquiera de ellos aparece otra lista con sus opciones de instalación. Para agregar o quitar componentes a la instalación se pincha sobre el pequeño recuadro situado a la izquierda de cada uno de los nombres.

El nombre del primer componente principal es *4th Dimension* y representa al programa en sí, por lo que deberá estar seleccionado. El segundo componente, *Help Files*, hace referencia a los ficheros de ayuda y se recomienda instalarlo en su totalidad. El siguiente es *tools* (utilidades) y permite instalar el lector de documentos Acrobat Reader, el contenedor de utilidades OLE (sólo para Windows 95 y Windows NT) y el programa *Customizer Plus* para personalizar las aplicaciones creadas con 4D. El lector de documentos es imprescindible para visualizar la documentación incluida en el CD-ROM. La cuarta opción sirve para



En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

PROGRAMACIÓN EN 3D

P *Hola, estoy leyendo su curso de Visual C++ 4.0 y veo que este paquete es muy completo para programar en Windows 95 o NT. Pero Windows 3.x ha tenido siempre una espina clavada que ha sido los juegos. Mi pregunta es la siguiente ¿qué opciones hay en el paquete de Visual C++ 4.0 para la programación de juegos en 3D? Otra cuestión, al ser Windows 95 un sistema operativo de 32 bits tiene muchas ventajas sobre el DOS, pero ¿son todas estas ventajas suficientes para poder realizar un juego de la calidad de XWING o de DOOM, sin precisar de un gran ordenador, específicamente para Windows 95, o por el contrario nos queda mucho tiempo con juegos para DOS?*

Cristóbal Seijo Bar
(Alicante)

R En el paquete de Visual C++ 4.0 no hay opciones específicas para programar juegos en 3D, aunque siempre puedes cargarte de paciencia y crear tu propio motor 3D. Sin embargo, en el momento de edición del número que tienes en tus manos Microsoft está dando los últimos toques al conjunto de servicios API para gráficos en tres dimensiones Direct3D. Forma parte de la tecnología DirectX de Microsoft, junto con DirectDraw, DirectSound, DirectInput y DirectPlay, que proporcionan una serie de servicios independientes del dispositivo para la creación de aplicaciones de alto rendimiento. Direct3D proporciona servicios como modelado Flat y Gouraud, mapas de texturas, transparencia, deformación de objetos en tiempo real y soporte de materiales, entre otros muchos. La versión definitiva de Direct3D estará al alcance de los desarrolladores como parte del SDK de DirectX 2.0 a través de la red Microsoft Developer Network.

Por otro lado, decirte que la potencia del ordenador ya no tiene tanta importancia. El estándar actual del mercado es el Pentium con 16 megas de RAM, un equipo capaz de ejecutar juegos bajo Windows 95 sin problemas. Muchas de las casas dedicadas al desarrollo de juegos ya se han pasado a esta plataforma y otras muchas lo están haciendo progresivamente. Lo cierto es que al DOS no le queda mucho tiempo de vida, y por tanto, a los juegos basados en DOS tampoco.

ACCESO A MULTIMEDIA

P *Me gustaría que me resolvieran un pequeño problema que tengo a la hora de programar para Windows. Tengo un compilador algo antiguo. Cuando me hice con él, no estaba tan de moda la multimedia, por lo que las librerías que posee no tienen acceso al vídeo, sonido, etc. Sé que en las DLLs están las instrucciones para acceder al mundo multimedia, por ejemplo en el archivo MMSYSTEM.DLL. Mi pregunta es la siguiente, ¿podría yo acceder a estas instrucciones sin el archivo cabecera (MMSYSTEM.H)? Si es así, ¿Cómo se podría hacer?*

Francisco G. Pérez Sánchez
(Berriozar / Navarra)

R Lo fundamental para acceder desde C o C++ a estas librerías es conocer el prototipo de las funciones que se van a utilizar. En realidad lo que se hace es llamar a una función con unos parámetros determinados para que realice alguna tarea y, llegado el caso, devuelva información sobre la misma. Por tanto, no necesita utilizar el archivo de cabecera MMSYSTEM.H, aunque es imprescindible que conozca las funciones que le proporciona dicha librería, y el prototipo de las funciones.

Si dispone de información sobre los prototipos la librería MMSYSTEM.DLL puede crear un archivo de cabecera propio con las funciones que necesita, incluirlo en su programa y realizar las llamadas pertinentes.

PREGUNTAS VARIAS

P *Hola, ante todo quisiera felicitarles por la revista que es de muy buena calidad. Tengo, entre otras, las siguientes dudas que espero puedan ayudarme a dilucidar: ¿Cómo puedo crear un motor Hypermedia en C++? ¿Hay algún producto comercial que me pueda ayudar?.*

Por otro lado, ¿Existe algún compilador de HTML? Mi idea es convertir un documento HTML en un ejecutable, empaquetando todos los archivos en una base de datos, con sus links, como si fuera una mini aplicación multimedia.

Alejandro Malbarez
(Buenos Aires / Argentina)

R Nos alegramos mucho de saber que también hay mucha gente interesada en el tema de la programación en Sudamérica. Vamos allá:

La creación de un motor hypermedia en C++ puede realizarse con cualquiera de los grandes compiladores de ese lenguaje disponibles actualmente en el mercado (Visual C++, Borland C++, Symantec C++). Estos compiladores proporcionan un gran número de librerías y utilidades idóneas para la creación de aplicaciones multimedia. El desarrollo de un motor hypermedia sería sólo cuestión de tiempo.

Que nosotros sepamos no existe ningún compilador de HTML como tal. Sin embargo sí tenemos conocimiento de programas capaces de crear una base de datos con lugares web y que permiten visualizarlos en modo local con cualquier navegador. Quizás le sirva el programa Grab-a-Site, del grupo "The ForeFront Group". Si tiene conexión a Internet, lo puede encontrar en el URL http ://www.bluesquirrel.com

ACCESO AL ALTAVOZ DESDE ENSAMBLADOR

P *Hace poco me decidí a aprender ensamblador y ahora estoy haciendo mis primeros pinitos con este lenguaje. Después de hacer varios programas pequeños me he animado a hacer una utilidad de captura de pantallas, pero tengo un problema. Necesito que cada vez que se grabe una pantalla correctamente el ordenador emita un sonido y si hubo problemas que emita otro sonido diferente avisando del error. No encuentro información sobre cómo utilizar el altavoz del PC desde ensamblador. ¿Podríais ayudarme?*

Miguel Angel Moreno Morales
(Tres Cantos / Madrid)

R La explicación detallada de cómo acceder al altavoz de nuestro querido PC desde lenguaje ensamblador excede el espacio del que disponemos. Te explicamos brevemente:

Se puede acceder de dos formas: directamente o mediante la conexión al temporizador número dos. Esta última es la más fiable y la más utilizada. Para ello, en primer lugar hay que programar el PIT (Programmable Interval Timer) para que genere la onda cuadrada que necesitamos. Se utilizan las siguientes instrucciones:

```
MOV AL, 0B6h
OUT 43h, AL
```

Con esto indicamos al PIT que vamos a acceder al segundo temporizador con escritura de ambos bytes del contador, y que debe generar una onda periódica cuadrada.

A continuación es necesario mandar al PIT el contador, un número de 16 bits que servirá para generar la onda cuadrada. En primer lugar se manda al puerto 42h el byte menos significativo y justo después el más significativo. Se haría de la siguiente forma:

```
MOV AL, xxh
OUT 42h, AL
MOV AL, xxh
OUT 42h, AL
```

Ya sólo queda conectar el altavoz al temporizador número dos. Esto se consigue activando los bits 0 y 1 del puerto del altavoz. Las instrucciones para ello son:

```
IN AL, 61h
OR AL, 3
OUT 61h, AL
```

Ahora el altavoz sonará indefinidamente hasta que lo paremos. Para apagar el altavoz se realiza la operación inversa a la anterior:

```
IN AL, 61h
AND AL, 0FCh
OUT 61h, AL
```

Para controlar el tiempo de duración de la nota se puede leer el contador de ticks con la interrupción 1Ah, o se puede colgar de la interrupción 8 una pequeña rutina que lleve el control del tiempo transcurrido.

ACERCA DEL CURSO DE COMPILADORES

P *Hola, amigos de SOLO PROGRAMADORES. He seguido el curso de creación de un compilador desde que comenzó, y estoy intentando crear un compilador para generar aplicaciones informativas en las que se tiene que mostrar texto con referencias cruzadas e imágenes en movimiento, y quisiera saber cómo puedo generar código que acceda a la pantalla y a ficheros, ya que desconozco el lenguaje ensamblador y no sé cómo crear nuevas cadenas de código en este lenguaje.*

Victor Manuel Hernández
(Zaragoza)

R No es necesaria la creación de nuevas cadenas de código máquina para dotar al compilador de Letra de funciones gráficas o de manejo de ficheros, simplemente se tienen que definir nuevas funciones a las que puedan acceder los programas del lenguaje, y dentro de estas funciones, que se pueden programar en C, se realizarán los accesos a ficheros o a pantalla. Es decir, se pueden crear nuevas funciones del tipo abrir_fichero( ), put_image( ), etc., implementadas en el código de cabecera de los programas compilados. Además es posible pasar muchas de las funciones de C a Letra, siempre que se puedan llamar empleando únicamente enteros de 16 bits con signo, ya que éste es el único tipo de variables soportado por el compilador que se ha creado en esta serie.



Servidores de SOFTWARE de IBM.

Para salir a escena es imprescindible la pieza central.

Presentamos los Servidores de Software de IBM. La pieza central de su sistema cliente/servidor.

Su red tiene clientes y tiene servidores. Pero para diseñar el sistema cliente / servidor más eficaz, hay algo imprescindible que lo conecta todo. Aquí es donde entran en acción los nuevos Servidores de Software de IBM. Un total de siete servidores diseñados para conseguir que cada solución específica de cliente/servidor sea la correcta. Funcionan por separado o conjuntamente y están basados en tecnologías IBM y Lotus rigurosamente fiables y probadas en empresas del mundo entero. Los Servidores de Software de IBM están pensados para funcionar en la plataforma que usted elija: OS/2, Windows NT o AIX. Ahora, según vaya exigiendo más a su red, asegúrese de que cuenta con el software apropiado para salir a escena.

Para recibir gratuitamente el CD de presentación de los Servidores de Software de IBM, llámenos al

☎ 900 99 45 01

(7 h. a 23 h. de lunes a viernes).

Lotus Notes
Una herramienta excepcional para el trabajo en grupo.

Servidor de Base de Datos
Ayuda a compartir datos entre empresas y grupos de trabajo.

Servidor de Conexión a Internet
Una base sólida para hacer negocios a través de Internet.

Servidor de Comunicaciones
Un potente sistema de comunicaciones multiprotocolo.

Servidor de Transacciones
Una infraestructura segura para aplicaciones esenciales.

Servidor de Directorios y Seguridad
Gestiona usuarios y recursos de toda la empresa desde un solo punto.

Servidor de Gestión de Sistemas
Gestiona los recursos de hardware y software de la red.

Visítenos en Internet:
www.software.ibm.com/info/ad210/



Soluciones para nuestro pequeño mundo